# NUMERICAL METHODS KIT

FOR MATLAB, SCILAB AND OCTAVE USERS

**Rohan Verma**

MSc (Physics), Indian Institute of Technology Kanpur, India

BSc (Physics), University of Delhi, India

# Preface

I take great pleasure in presenting to the readers this book entitled "Numerical Methods Kit". The book has been designed for Science, Engineering, Mathematics and Statistics students.

A look at the contents of the book will give the reader a clear idea of the variety of numerical methods discussed and analysed.

The book has been written in a concise and lucid style with proper explanation of Mathematics involved in each method.

Each method is explained with solved examples, computer programs and their results like a screenshot of the graphic window and console window.

The careful organisation of figures, solved examples, codes, graphic window and console window help the students grasp quickly.

Despite careful editing, there is zero probability that this book is error-free. If anything looks amiss, please send that part to my email id given below.

Email: rohanv.i341@gmail.com Rohan Verma

# CONTENTS

# CHAPTER **1**

# Solutions of Algebraic and Transcendental Equations

$f(x) = 0$ is called equation and the values of $x$ for which this equation satisfied is called roots of the equation.

Algebraic Equation - If $f(x)$ is purely polynomial. E.g. $x^3 - x + 1 = 0$

Transcendental Equations - If $f(x)$ containing logarithm, trigonometric, exponential function, etc. E.g. $xe^x - cos\, x = 0$

## 1. 1 Bisection Method

The bisection method applies to a continuous function. Let our equation is $f(x) = 0$. Here (Figure 1.1) we are going to consider two values of $x$, $a$ and $b$ such that $f(a) < 0$ and $f(b) > 0$. Clearly, we can see in Figure 1.1 that the root must lie somewhere between $a$ and $b$.

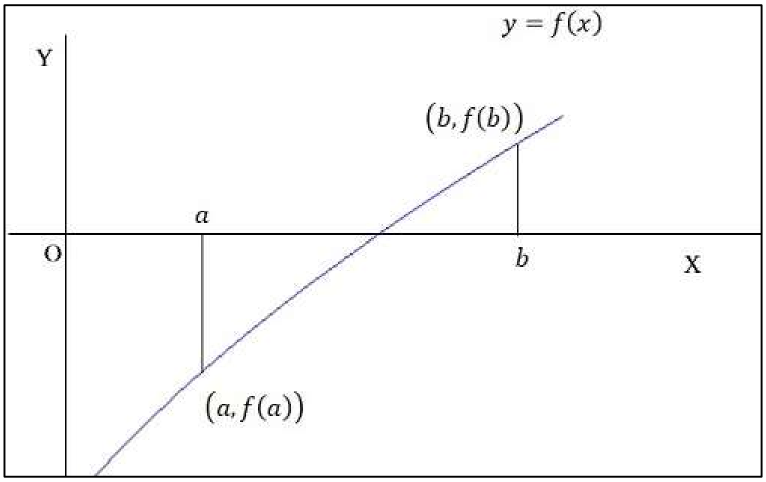


**Figure 1. 1**

Take $c = \frac{a+b}{2}$

There could be three possibilities.

(a) $f(c) > 0$

replace $b$ with $c$

(b) $f(c) < 0$

replace $a$ with $c$

(c) $f(c) = 0$

$c$ is the root of equation $f(x) = 0$. We are replacing $a$ or $b$ with $c$ to make a smaller interval for finding the root. In this way, we keep bisecting the interval for finding the root. Eventually, the interval will coincide with the $c$ for which $f(c) = 0$.

**Working rule**

(a) Find $a$ and $b$ such that $f(a) < 0$ and $f(b) > 0$ or $\{f(a)f(b)\} < 0$.
(b) Find the first approximate root using.

$$x_1 = \frac{a+b}{2}$$

Now calculate $f(x_1)$ and Examine its sign. If $f(x_1) < 0$ it implies that root lies between $x_1$ and $b$. 2nd approximate root is given by $x_2 = \frac{x_1+b}{2}$.
If $f(x_1) > 0$ it implies that root lies between $a$ and $x_1$.Then 2nd approximate root is given by $x_2 = \frac{a+x_1}{2}$. Calculate $f(x_2)$. Repeat the (b) step until the required accuracy of root or $f(x_n) \to 0$.

**Example 1. 1** Find the 5th approximate real root of the equation $x^2 - 1 = 0$ using the Bisection Method.
Let $f(x) = x^2 - 1 = 0$
take $a = 0\ .8$ and $b = 1.1$

$$f(0.8) = -0.36 < 0$$
$$f(1.1) = 0.21 > 0$$

First approximate root is

$$x_1 = \frac{0.8 + 1.1}{2} = 0.95$$
$$f(0.95) = -0.0975$$

New value of $a = 0.95$
Second approximate root is

$$x_2 = \frac{0.95 + 1.1}{2} = 1.025$$
$$f(1.025) = 0.050625$$

New value of $b = 1.025$
Third approximate root is

$$x_3 = \frac{0.95 + 1.025}{2} = 0 \cdot 9875$$
$$f(0.9875) = -0.0248437$$

New value of $a = 0.9875$

Fourth approximate root is

$$x_4 = \frac{0.9875 + 1.025}{2} = 1.00625$$

$$f(1.00625) = 0.0125391$$

New value of $b = 1.00625$

Fifth approximate root is

$$x_5 = \frac{0.9875 - 1 \cdot 00625}{2} = 0.996875$$

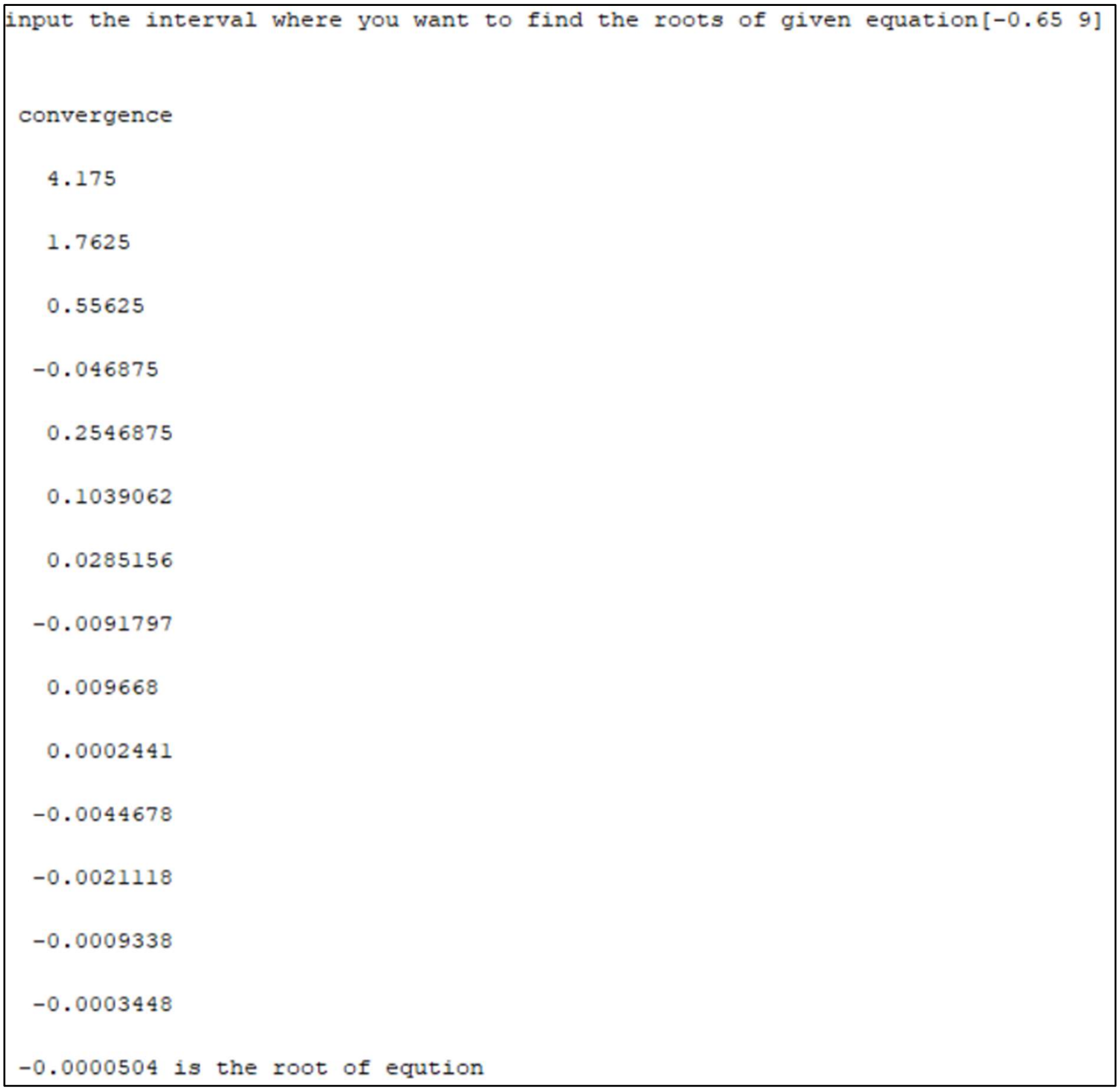
$$f(0.996875) = -0.0062402$$

In the previous example, we can see that $x_n \to r$, where $r$ is the root of equation $f(x) = 0$.

**Scilab Code 1.1**

```
//bisection method
clc
clear
function z=f(x)
   z=x^3
endfunction
n=input("input the interval where you want to find the roots of given
equation")
a=n(1)
b=n(2)
c=(a+b)/2
tol=0.00000000001
if f(a)*f(b)<0 then

disp("convergence")
while abs(f(c))>tol
   if f(a)*f(c)<0 then
      b=c
   else
      a=c
   end

   disp(c)
   c=(a+b)/2
end
disp(""+string(c)+" is the root of eqution")
else
   disp('root does not exist in this interval')
   end
```

**Console Window 1. 1**



```
input the interval where you want to find the roots of given equation[-0.65 9]


 convergence

   4.175

   1.7625

   0.55625

  -0.046875

   0.2546875

   0.1039062

   0.0285156

  -0.0091797

   0.009668

   0.0002441

  -0.0044678

  -0.0021118

  -0.0009338

  -0.0003448

 -0.0000504 is the root of eqution
```

## 1. 2 Newton-Raphson Method

The Newton - Raphson Method applies to continuous and differentiable function. Let our equation is $f(x) = 0$. In Newton - Raphson Method First we need to consider some initial guess let take $x_0$, and then we draw a tangent at $\left(x_0, f(x_0)\right)$ (Figure 1.2). This tangent intersects at x-axis at $x_1$. from Figure 1.2 we can see that $x_1$ is closer to root. To get closer values $(x_1, x_2 \dots x_n)$, we repeat the same process for $x_1$ and so on until $x_n \to r$, where $r$ is the root of equation $f(x) = 0$. The rate of convergence of the Newton - Raphson Method is faster than the Bisection method.

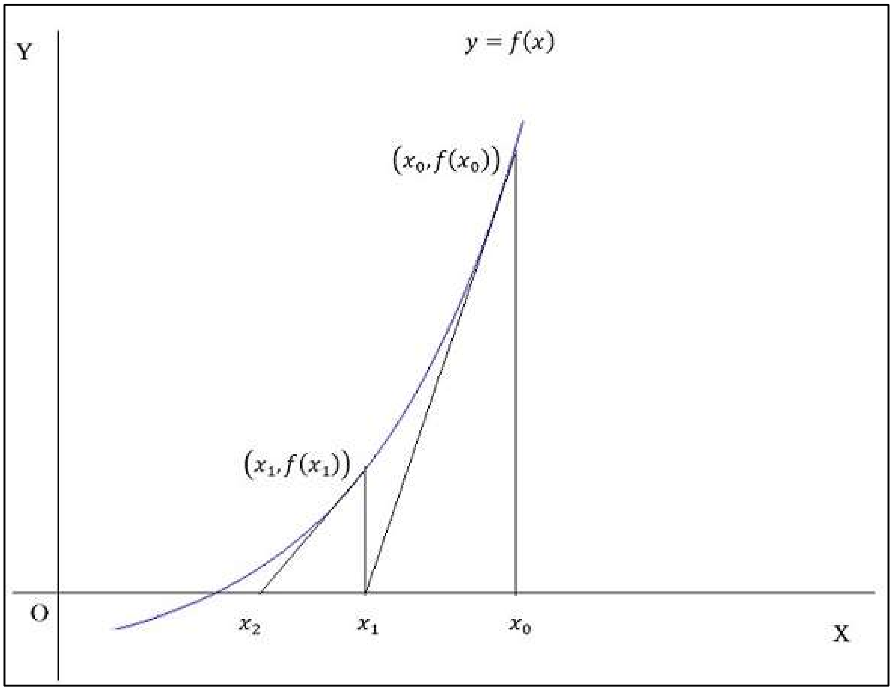


**Figure 1. 2**

**Working rule**

(a) Consider equation $f(x) = 0$ to find root by Newton - Raphson method

And take initial guess root $x_0$.

(b) Find 1st approximate root $x_1$ by constructing a tangent equation at $(x_0, f(x_0))$ and putting $y = 0$.

$$y - f(x_0) = f'(x_0)(x - x_0)$$

Putting $y = 0$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

(c) 2nd approximate root

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

(d) repeat the (b) step until $x_n \to r$, where $r$ is the root of equation $f(x) = 0$.

**General Formula**

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}, n = 1,2,3, \dots \tag{1.1}$$

**Example 1. 2** Find the $6^{th}$ approximate real root of the equation $x^2 - 2x + 1 = 0$ by Newton - Raphson Method.

Let $f(x) = x^2 - 2x + 1$

$f'(x) = 2x - 2$

Consider initial guess 3

1st approximate root

$$x_1 = 3 - \frac{f(3)}{f'(3)}$$

$$x_1 = 2$$

$$f(2) = 1$$

2nd approximate root

$$x_2 = 2 - \frac{f(2)}{f'(2)}$$

$$x_2 = 1.5$$

$$f(1.5) = 0.25$$

3rd approximate root

$$x_3 = 1.5 - \frac{f(1.5)}{f'(1.5)}$$

$$x_3 = 1.25$$

$$f(1.25) = 0.0625$$

4th approximate root

$$x_4 = 1.25 - \frac{f(1.25)}{f'(1.25)}$$

$$x_4 = 1.125$$

$$f(1.125) = 0.015625$$

5th approximate root

$$x_5 = 1.125 - \frac{f(1.125)}{f'(1.125)}$$

$$x_5 = 1.0625$$

$$f(1.0625) = 0.0039062$$

6th approximate root

$$x_6 = 1.0625 - \frac{f(1.0625)}{f'(1.0625)}$$

$$x_6 = 1.03125$$

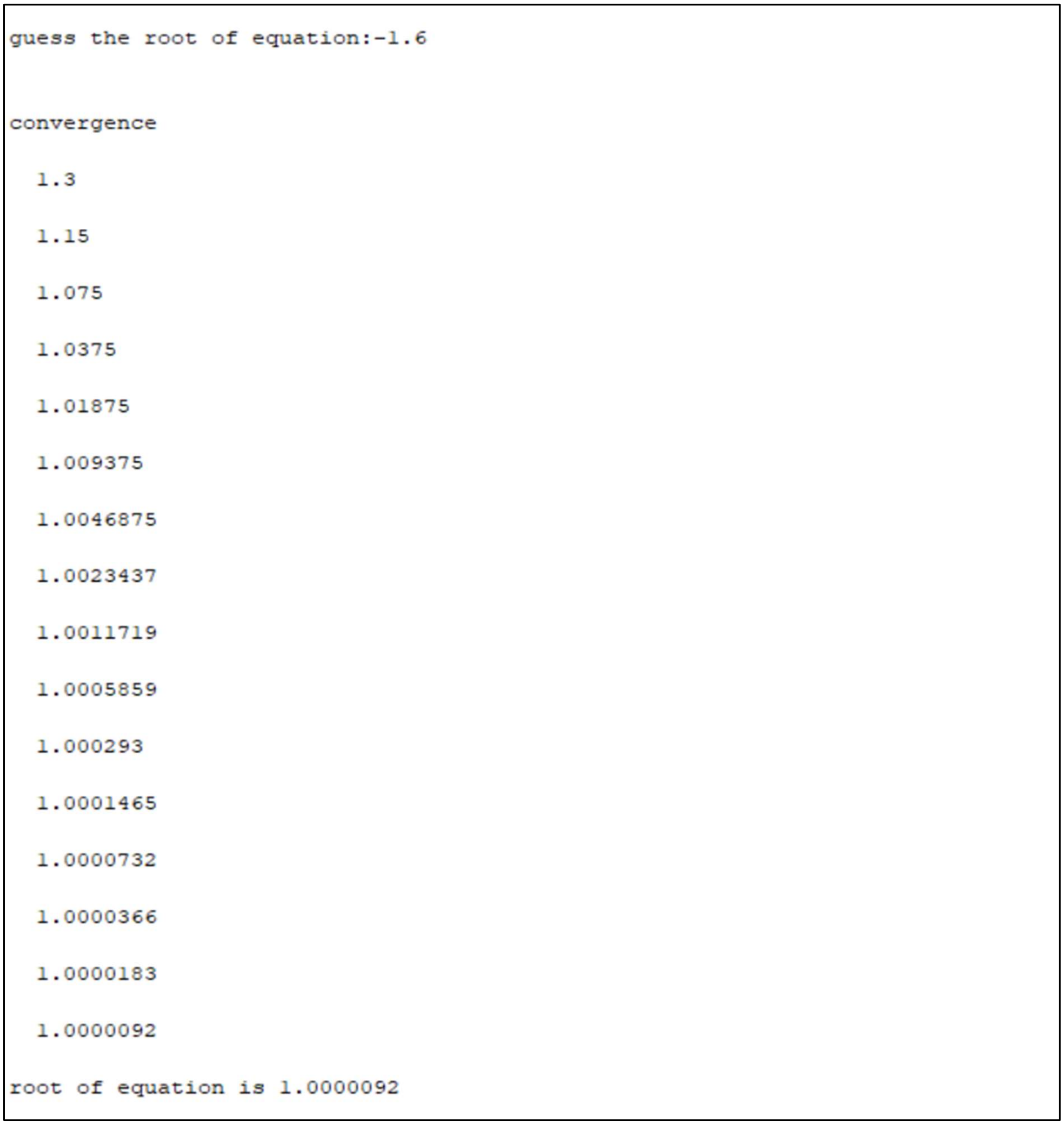
$$f(1.03125) = 0.0009766$$

6th approximate root is 1.03125 for equation $x^2 - 2x + 1 = 0$.

### Scilab Code 1.2

```
//Newton-Raphson Method
clc
clear
x0=input("guess the root of equation:-")
function z=f(x)
   z=x^2-2*x+1
endfunction
disp("convergence")
tol=0.0000000001
while abs(f(x0))>tol
   x1=x0-f(x0)/numderivative(f,x0)
   disp(x1)
   x0=x1
end
disp("root of equation is "+string(x0)+"")
```

**Console Window 1.2**



```
guess the root of equation:-1.6


convergence

  1.3

  1.15

  1.075

  1.0375

  1.01875

  1.009375

  1.0046875

  1.0023437

  1.0011719

  1.0005859

  1.000293

  1.0001465

  1.0000732

  1.0000366

  1.0000183

  1.0000092

root of equation is 1.0000092
```

## 1.3 Regula-Falsi Method

Let our equation is $f(x) = 0$. First, we need to consider a root-finding interval $(a, b)$. Such that $f(a) < 0$ and $f(b) > 0$ or $\{f(a)f(b)\} < 0$. It is quite obvious that root must lie somewhere between $a$ and $b$ (see Figure 1.3). Here we can see that it is looking very similar to the Bisection Method till now. But in the Regula Falsi Method instead of the bisecting interval, we draw a line segment joining $(a, f(a))$ and $(b, f(b))$ see Figure 1.3, this line segment intersects at x-axis at $x_1$ .This $x_1$ is closer to the root of equation $f(x) = 0$, we can say that $x_1$ is our first approximate root of the equation.

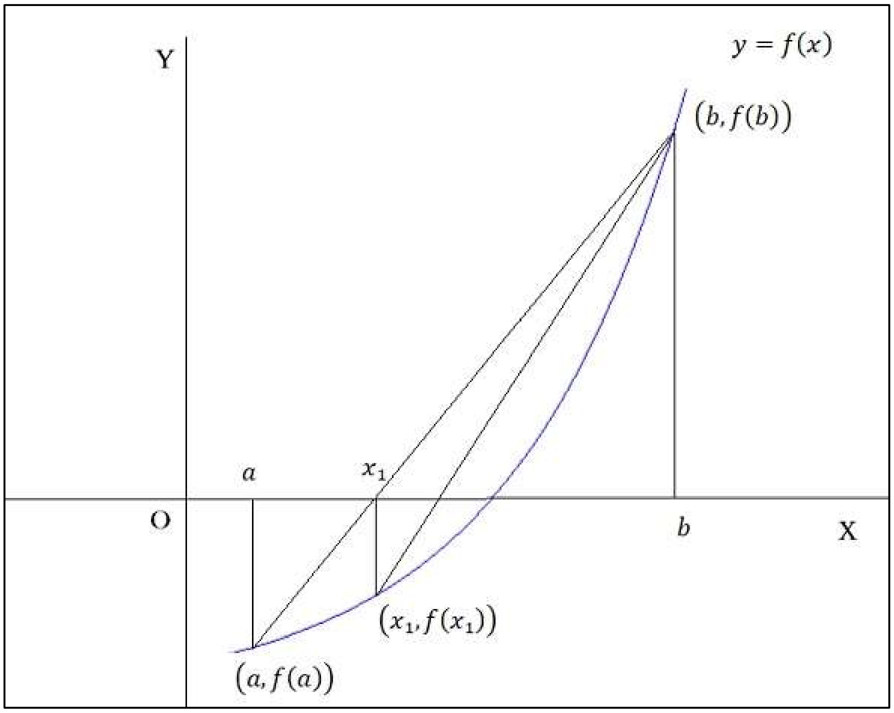


**Figure 1.3**

Now for 2nd approximate root, we check the sign of $f(x_1)$

If $f(x_1\ ) > 0$ replace $b$ with $x_1$.

If $f(x_1\ ) < 0$ replace $a$ with $x_1$.

Keep doing this process until $x_n \to r$, where $r$ is the root of equation $f(x) = 0$.

**Working rule**

(a) Let $f(x) = 0$ be the equation to solve. Take $a$ and $b$ such that $f(a) < 0$ and $f(b) > 0$.

(b) Find 1st approximate root $x_1$ by constructing equation of line segment Joining $(a, f(a))$ and $(b, f(b))$ and putting $y = 0$.

$$y - f(a) = \frac{f(b) - f(a)}{b - a}(x - a)$$

Putting $y = 0$

$$x_1 = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{1.2}$$

(c) Check the sign of $f(x_1)$.

If $f(x_1) > 0$ replace $b$ with $x_1$

2nd approximate root

$$x_2 = \frac{af(x_1) - x_1 f(a)}{f(x_1) - f(a)}$$

If $f(x_1) < 0$ replace $a$ with $x_1$

2nd approximate root

$$x_2 = \frac{x_1 f(b) - bf(x_1)}{f(b) - f(x_1)}$$

(c) repeat the third step until convergence reached.

**Example 1.3** Find the 7th approximate root of $xe^x = cos(x)$ by Regula Falsi method.

Let $f(x) = xe^x - cos(x) = 0$ and $a = 0, b = 1$

$f(0) = -1 < 0$ and $f(1) = 2.1779795 > 0$

1st approximate root

$$x_1 = \frac{0f(1) - 1f(0)}{f(1) - f(0)}$$

$$x_1 = 0.3146653$$

$$f(0.3146653) = -0.5198712 < 0$$

New value of $a = 0.3146653$

2nd approximate root

$$x_2 = \frac{0.3146653f(1) - 1f(0.3146653)}{f(1) - f(0.3146653)}$$

$$x_2 = 0.4467281$$

$$f(0.4467281) = -0.2035448 < 0$$

New value of $a = 0.4467281$

3rd approximate root

$$x_3 = \frac{0.4467281\ (1) - 1f(0.4467281\ )}{f(1) - f(0.4467281\ )}$$

$$x_3 = 0.4940153$$

$$f(0.4940153) = -0.0708023 < 0$$

New value of $a = 0.4940153$

4th approximate root

$$x_4 = \frac{0.4940153\ \ f(1) - 1f(0.4940153\ \ )}{f(1) - f(0.4940153\ )}$$

$$x_4 = 0.5099461$$

$$f(0.5099461) = -0.0236077 < 0$$

New value of $a = 0.5099461$

5th approximate root

$$x_5 = \frac{0.5099461\ \ f(1) - 1f(0.5099461\ \ )}{f(1) - f(0.5099461\ )}$$

$$x_5 = 0.515201$$

$$f(0.515201) = -0.0077601 < 0$$

New value of $a = 0.515201$

6th approximate root

$$x_6 = \frac{0.515201\ \ f(1) - 1f(0.515201\ \ )}{f(1) - f(0.515201)}$$

$$x_6 = 0.5169222$$

$$f(0.5169222) = -0.0025389 < 0$$

New value of $a = 0.5169222$

7th approximate root

$$x_7 = \frac{0.5169222\ \ f(1) - 1f(0.5169222\ \ )}{f(1) - f(0.5169222)}$$

$$x_7 = 0.5174847$$

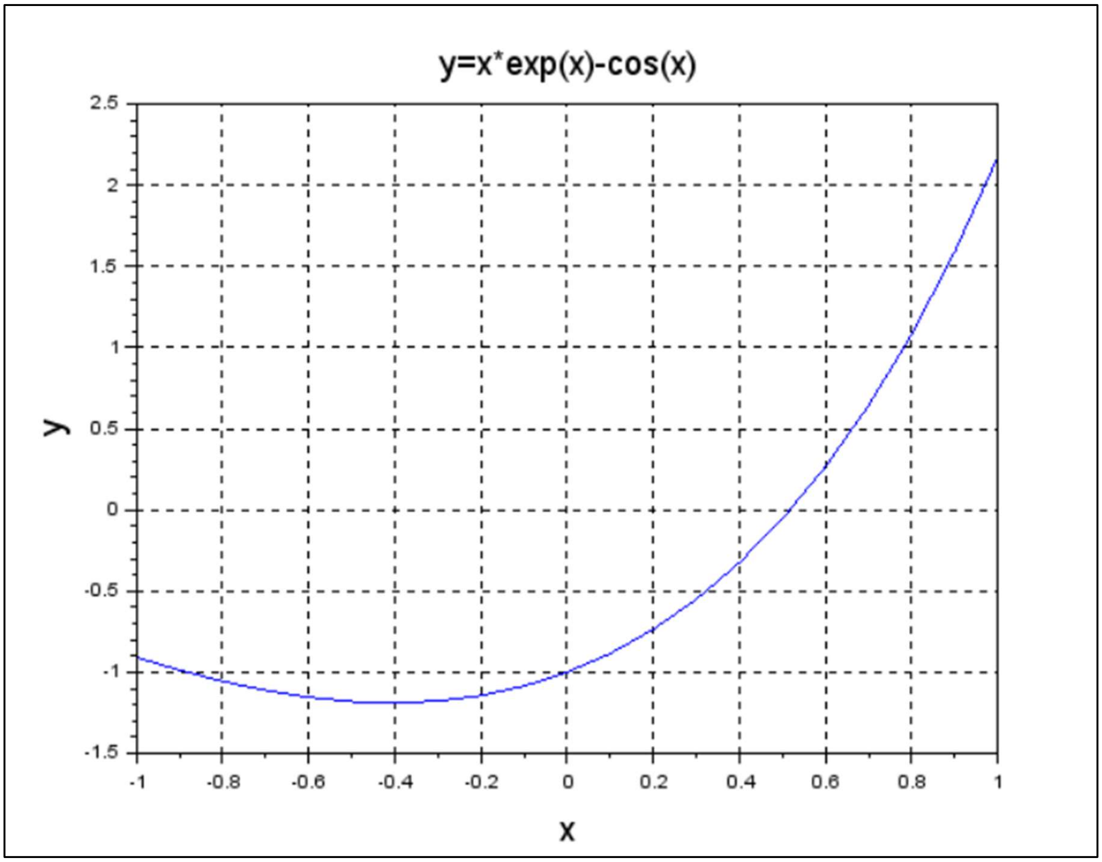
$$f(0.5174847) = -0.0008294$$

7th approximate root is 0.5174847 for equation $xe^{x} - cos(x) = 0$.

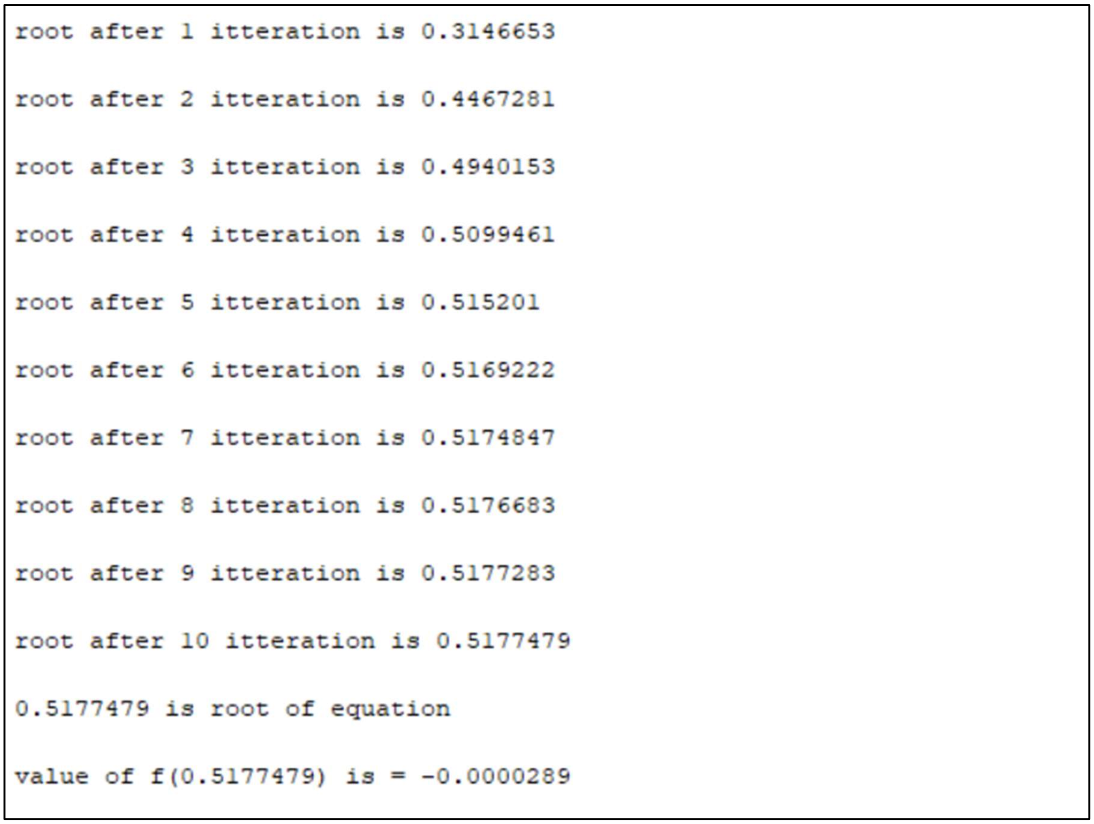
### Scilab Code 1.3

```
//Regula-Falsi Method
clc
clear
clf
x=[-1:0.1:1]
function z=f(x)
   z=x*exp(x)-cos(x)
endfunction
plot(x,f,'c')
xgrid(3)
xlabel("x","Fontsize",4)
ylabel("y","Fontsize",4)
title("y=x*exp(x)-cos(x)","Fontsize",4)
a=0
b=1
for i=1:10
   x1=(a*f(b)-b*f(a))/(f(b)-f(a))
   disp('root after '+string(i)+' iteration is '+string(x1)+'')
   if f(x1)< 0 then
      a=x1
   end
   if f(x1)> 0 then
      b=x1
   end
    if f(x1)==0 then
      break
      end
end
disp(''+string(x1)+' is root of equation' )
disp('value of f('+string(x1)+') is = '+string(f(x1))+'')
```

**Graphic Window 1. 1**



**Console Window 1. 3**



```
root after 1 itteration is 0.3146653

root after 2 itteration is 0.4467281

root after 3 itteration is 0.4940153

root after 4 itteration is 0.5099461

root after 5 itteration is 0.515201

root after 6 itteration is 0.5169222

root after 7 itteration is 0.5174847

root after 8 itteration is 0.5176683

root after 9 itteration is 0.5177283

root after 10 itteration is 0.5177479

0.5177479 is root of equation

value of f(0.5177479) is = -0.0000289
```

## 1. 4 Secant Method

Secant method is an improved form of Regula Falsi method. In this method, we don't need to check the sign of the approximate root. Like we have done in the Regula Falsi method. Let our equation is $f(x) = 0$ with $f(x_0) < 0$ and $f(x_1) > 0$. So that mean root must lie between $x_0$ and $x_1$. First approximate root $x_2$ is given by

$$x_2 = \frac{x_0 f(x_1) - f(x_0) x_1}{f(x_1) - f(x_0)}$$

Similarly, Second approximate root $x_3$ is given without checking the sing of $f(x_2)$.

$$x_3 = \frac{x_1 f(x_2) - f(x_1) x_2}{f(x_2) - f(x_1)}$$

**General Formula**

$$x_{n+1} = \frac{x_{n-1} f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}, n \geq 1 \qquad (1.3)$$

Keep finding $n$th approximate root until $x_n \to r$ where $r$ is the root of equation $f(x) = 0$.

**Example 1.4** Find the 7th approximate real root of the equation $2^x - x^2 = 0$ in interval $-1 \leq x \leq 1$ by secant method.

Let $f(x) = 2^x - x^2 = 0$ and $x_0 = -1, x_1 = 1$

$f(-1) = -0.5 < 0$ and $f(1) = 1 > 0$

1st approximate root

$$x_2 = \frac{-1f(1) - 1f(-1)}{f(1) - f(-1)}$$

$$x_2 = -0.3333333$$

$$f(-0.3333333) = 0.6825894$$

2nd approximate root

$$x_3 = \frac{1f(-0.3333333) - (-0.3333333)f(1)}{f(-0.3333333) - f(1)}$$

$$x_3 = -3.2006581$$

$$f(-3.2006581) = -10.135443$$

3rd approximate root

$$x_4 = \frac{-0.3333333\ f(-3.2006581) - (-3.2006581)f(-0.3333333\ )}{f(-3.2006581) - f(-0.3333333\ )}$$

$$x_4 = -0.514254$$

$$f(-0.514254) = 0.4356977$$

4th approximate root

$$x_5 = \frac{-3.2006581\ \ f(-0.514254) - (-0.514254)f(3.2006581\ \ )}{f(-0.5142541) - f(-3.2006581\ )}$$

$$x_5 = -0.6249762$$

$$f(-0.6249762) = 0.2578352$$

5th approximate root

$$x_6 = \frac{-0.514254\ \ f(-0.6249762) - f(-0.6249762)(-0.514254\ \ )}{f(-0.6249762) - f(-0.514254)}$$

$$x_6 = -0.7854827$$

$$f(-0.7854827) = -0.0368252$$

6th approximate root

$$x_7 = \frac{-0.6249762\ \ f(-0.7854827) - (-0.7854827)f(-0.6249762\ \ )}{f(-0.7854827) - f(-0.6249762)}$$

$$x_7 = -0.7654234$$

$$f(-0.7654234) = 0.0024078$$

7th approximate root

$$x_8 = \frac{-0.7854827\ \ f(-0.7654234) - (-0.7654234)f(-0.7854827\ \ )}{f(-0.7654234) - f(-0.7854827)}$$

$$x_8 = -0.7666544$$

$$f(-0.7666544) = 0.0000199$$

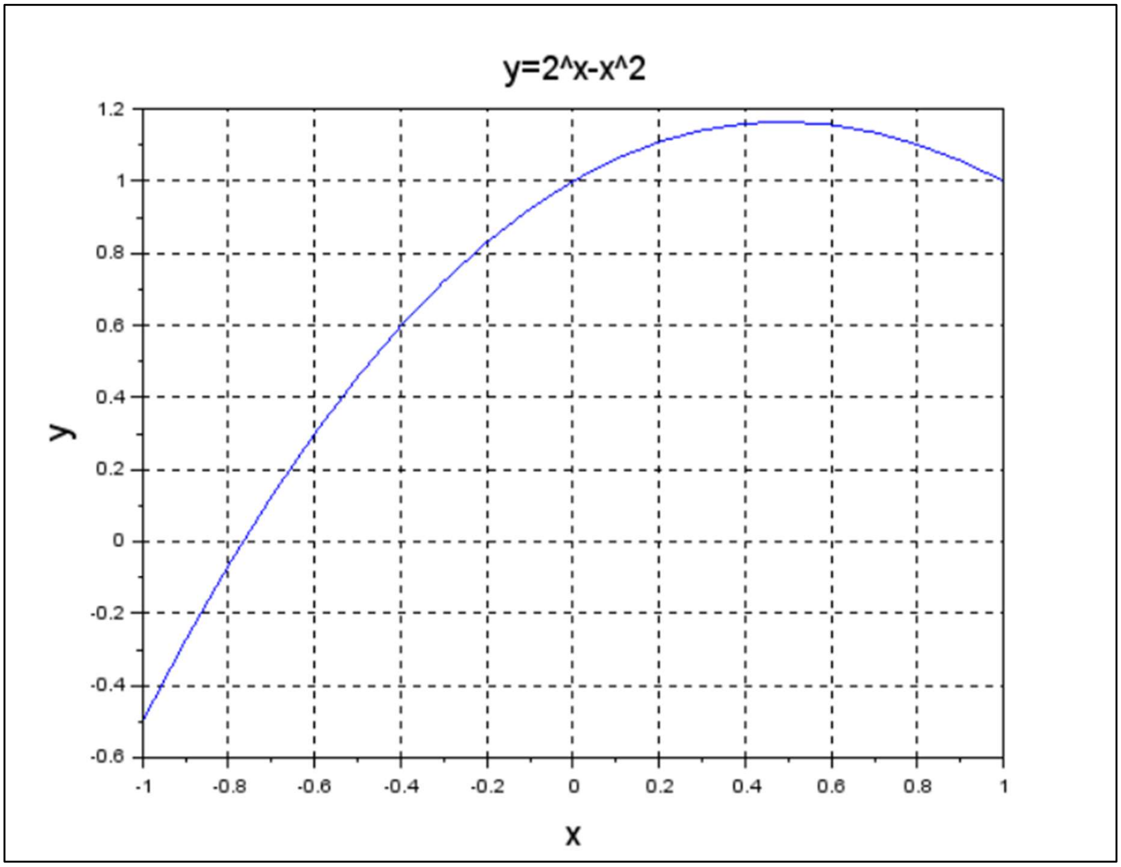
7th approximate root is $-0.7666544$ for equation $2^x - x^2 = 0$.

**Scilab Code 1.4**

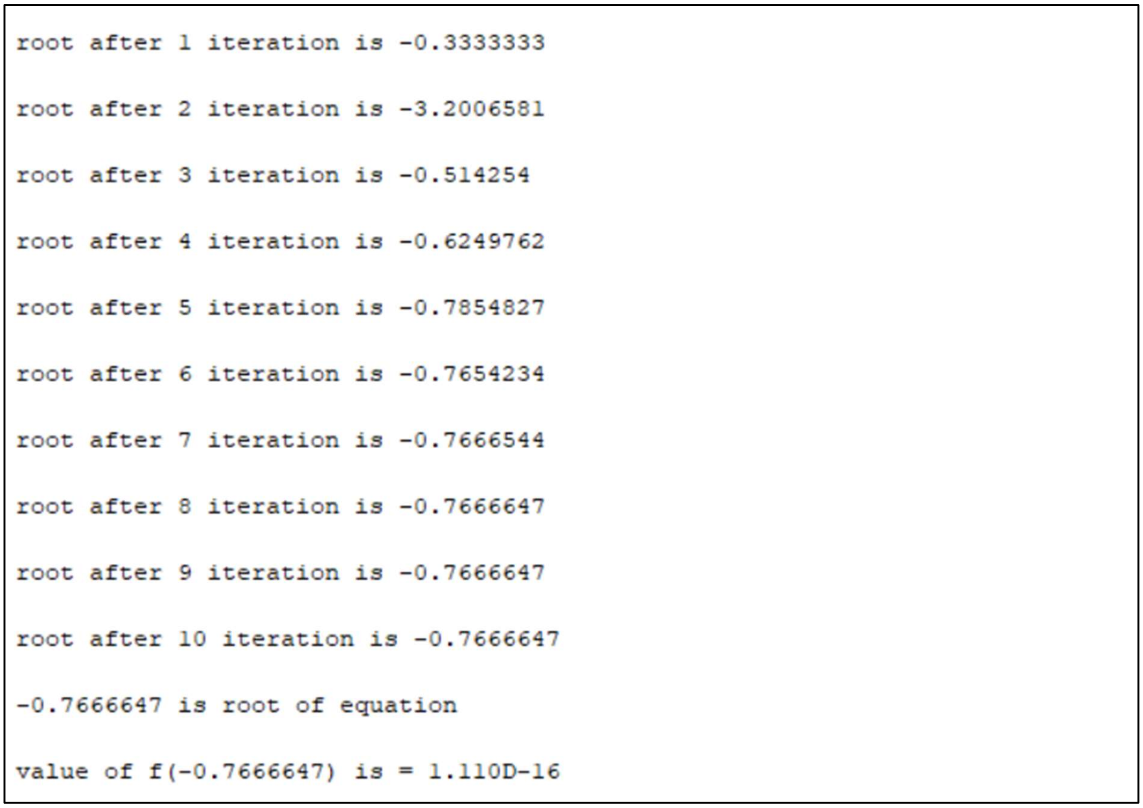
```
//Secant Method
clc
clear
clf
x=[-1:0.1:1]
function z=f(x)
   z=2^x-x^2
endfunction
plot(x,f,'c')
xgrid(3)
xlabel("x","Fontsize",4)
ylabel("y","Fontsize",4)
title("y=2^x-x^2","Fontsize",4)
a=-1
b=1
for i=1:10
   x1=(a*f(b)-b*f(a))/(f(b)-f(a))
   disp('root after '+string(i)+' iteration is '+string(x1)+'')
   disp(f(x1))
```

```
    a=b
    b=x1
     if f(x1)==0 then
        break
        end
end
disp(''+string(x1)+' is root of equation' )
disp('value of f('+string(x1)+') is = '+string(f(x1))+'')
```

**Graphic Window 1. 2**



**Console Window 1. 4**



```
root after 1 iteration is -0.3333333

root after 2 iteration is -3.2006581

root after 3 iteration is -0.514254

root after 4 iteration is -0.6249762

root after 5 iteration is -0.7854827

root after 6 iteration is -0.7654234

root after 7 iteration is -0.7666544

root after 8 iteration is -0.7666647

root after 9 iteration is -0.7666647

root after 10 iteration is -0.7666647

-0.7666647 is root of equation

value of f(-0.7666647) is = 1.110D-16
```

# CHAPTER 2

# Interpolation

Interpolation is the way of estimating function $f(x)$ for given data $(x_i, y_i)$ within the range. Whereas Extrapolation is about estimating the function outside the given data.

There are many methods for Interpolation and Extrapolation. A few of them are listed below.

(a) Newton's Gregory Forward Interpolation Method

(b) Newton's Gregory Backward Interpolation Method

(c) Lagrange's Interpolation Method

(d) Newton's Divided Difference Interpolation method

Newton's forward and backward interpolation method are used for constant step size ($|x_n - x_{n-1}| = constant$).

Lagrange and Newton's Divided Difference Interpolation method are used for varying step size.

It's good to use Newton's forward method for better estimation $f(x_i)$ when $x_i$ is closer to the first value of range and vice versa. It's ok to use Newton's Forward or Backward Interpolation Method to estimate $f(x_i)$. Where $x_i$ may lie inside or outside the range. Before getting into details of methods, first, we need to learn Finite Difference operators.

## 2. 1 Finite Difference Operator

For $y = f(x)$ given data $(x_0, x_1, x_2 \dots x_n)$ and corresponding $y_i = f(x_i)$ $(y_0, y_1, y_2 \dots y_n)$

such that $x_n = x_0 + nh$, where $n \geq 1$ and $h$ is the length of the interval which is constant.

**Shifting Operator (E)**

It defines as $Ef(x) = f(x+h)$ similarly $E^2 f(x) = E\big(Ef(x)\big) = E\big(f(x+h)\big) = f(x+2h)$, $E^{-2}f(x) = f(x-2h)$,$E^{\frac{-1}{2}}f(x) = f\left(x - \frac{h}{2}\right)$.

**General Formula**

$$E^n f(x) = f(x+nh) \tag{2.1}$$

**Forward Difference Operator ($\varDelta$)**

It defines as $\varDelta f(x) = f(x+h) - f(x)$.

$$\begin{aligned}\varDelta^2 f(x) &= \varDelta\big(\varDelta f(x)\big)\\ &= \varDelta\big(f(x+h) - f(x)\big)\\ &= \varDelta f(x+h) - \varDelta f(x)\\ &= f(x+2h) - f(x+h) - (\,f(x+h) - f(x))\\ &= f(x+2h) - 2f(x+h) + \ f(x)\end{aligned}$$

$$\begin{aligned}\varDelta y_0 &= \varDelta f(x_0)\\ &= f(x_0+h) - f(x_0)\\ &= f(x_1) - f(x_0)\\ \varDelta y_0 &= y_1 - y_0\end{aligned}$$

Similarly

$$\begin{aligned}\varDelta y_1 &= y_2 - y_1\\ \varDelta y_2 &= y_3 - y_2\end{aligned}$$

**General Formula**

$$\varDelta y_n = y_{n+1} - y_n \tag{2.2}$$

**Backward Difference Operator ($\nabla$)**

It defines as $\nabla f(x) = f(x) - f(x-h)$.

$$\begin{aligned}\nabla y_1 &= \nabla f(x_1)\\ &= f(x_1) - f(x_1 - h)\\ &= f(x_1) - f(x_0)\\ &= \nabla y_1 = y_1 - y_0\end{aligned}$$

**General Formula**

$$\nabla y_n = y_n - y_{n-1} \tag{2.3}$$

**The relation between Finite Difference Operator**

$$\varDelta = E - 1 \tag{2.4}$$

$$\nabla = 1 - E^{-1} \tag{2.5}$$

## 2. 2 Newton's Gregory Forward Interpolation Method

Let for $y = f(x)$ a given data point is $(y_0, y_1, y_2 \ldots . y_n)$, $(x_0, x_1, x_2 \ldots . x_n)$ . In this method we estimate $f(x_i)$ or $y_i$ where $x_i \in (x_0, x_n)$. Data points of the independent variable must be equally spaced ($|x_i - x_{i-1}| = constant$). So $x_n = x_0 + nh$ , $n = 0,1,2,3, \ldots$ and $h$ is some constant. Finding $y = f(x)$ for $x \in (x_0, x_n)$.

Let $x = x_0 + uh$ or $u = \frac{x-x_0}{h}$ (2.6)

$$E^u f(x) = f(x+uh)$$
$$E^u f(x_0) = f(x_0+uh)$$

Using (2.6)

$$y = f(x) = E^u y_0$$

Using (2.4)

$$y = (1+\Delta)^u y_0 \qquad (2.7)$$

Binomial expansion of $(1+\Delta)^u$ is

$$(1+\Delta)^u = 1 + \frac{u\Delta}{1!} + \frac{u(u-1)\Delta^2}{2!} + \frac{u(u-1)(u-2)\Delta^3}{3!} + \cdots \qquad (2.8)$$

From 2.7 and 2.8 equation

$$\boxed{y = y_0 + \frac{u\Delta y_0}{1!} + \frac{u(u-1)\Delta^2 y_0}{2!} + \frac{u(u-1)(u-2)\Delta^3 y_0}{3!} + \cdots} \qquad (2.9)$$

Our equation (2.9) is Newton's Forward Interpolation formula, where $u = \frac{x-x_0}{h}$.

**Formation of Forward difference table using $\boldsymbol{\Delta y_n = y_{n+1} - y_n}$.**

Start the calculations from the first data point.

| $x$ | $y$ | $\Delta y$ | $\Delta^2 y$ | $\Delta^3 y$ |
|---|---|---|---|---|
| $x_0$ | $y_0$ | $\Delta y_0 = y_1 - y_0$ | $\Delta^2 y_0 = \Delta y_1 - \Delta y_0$ | $\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 \Delta y_0$ |
| $x_1$ | $y_1$ | $\Delta y_1 = y_2 - y_1$ | $\Delta^2 y_1 = \Delta y_2 - \Delta y_1$ | |
| $x_2$ | $y_2$ | $\Delta y_2 = y_3 - y_2$ | | |
| $x_3$ | $y_3$ | | | |

**Table 2.1**

**Example 2.1** Find the cubic polynomial which takes the following values.

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $f(x)$ | 1 | 2 | 1 | 10 |

Find $f(4)$ and $f'(4)$.

The Forward difference table is given from Table 2.1

| $\boldsymbol{x}$ | $\boldsymbol{y}$ | $\boldsymbol{\Delta y}$ | $\boldsymbol{\Delta^2 y}$ | $\boldsymbol{\Delta^3 y}$ |
|---|---|---|---|---|
| 0 | 1 | $\Delta y_0$ =1 | $\Delta^2 y_0 = -2$ | $\Delta^3 y_0 = 12$ |
| 1 | 2 | $\Delta y_1 = -1$ | $\Delta^2 y_1 = 10$ | |
| 2 | 1 | $\Delta y_2 = 9$ | | |
| 3 | 10 | | | |

**Table 2.2**

Here $h = 1$ ,$x_0 = 0$

Using (2.6) ,(2.9) and Table 2.2

$u = x$ and $f(x) = 2x^3 - 7x^2 + 6x + 1$

$$f'(x) = 6x^2 - 14x + 6$$

$$f(4) = 41$$

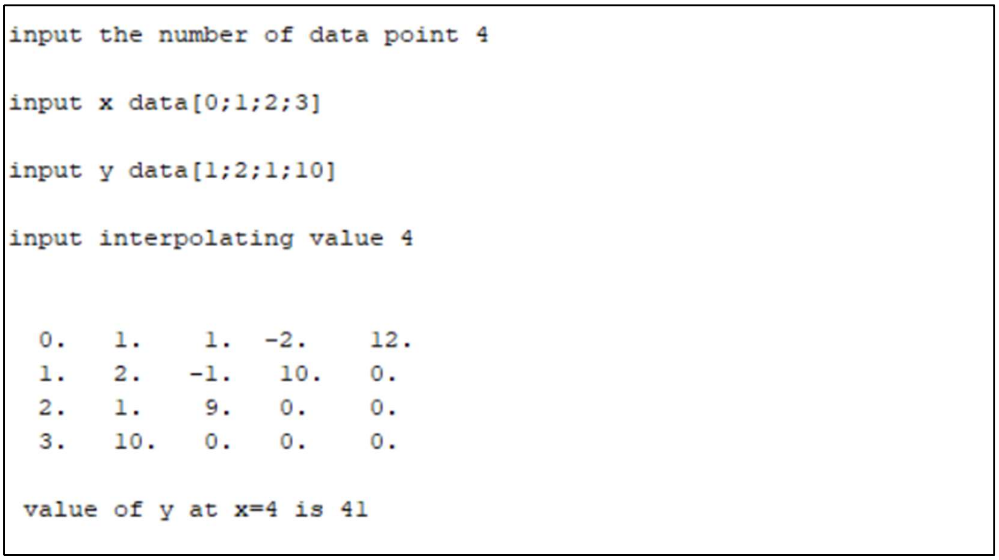
$$f'(4) = 46$$

**Scilab Code 2. 1**

```
//NEWTON'S FORWARD INTERPOLATION
clc
clear
//matrix u store table 2.1
n=input(' input the number of data point ')
u=zeros(n,n+1)
u(:,1)=input(' input x data')
u(:,2)=input(' input y data')
h=abs(u(2,1)-u(1,1))//step size
x=input(' input interpolating value ')

   a=(x-u(1,1))/h

for j=3:n+1//Formation of Forward difference table
   for i=1:(n-j+2)
      u(i,j)=u(i+1,j-1)-u(i,j-1)
end
end
disp(u)
//s variable is for summation of terms in equation (2.9)
```

```
s=u(1,2)
t=1
  for i=0:n-2
      t=(a-i)*t
      s=s+t*u(1,i+3)/factorial(i+1)
   end
 disp(' value of y at x='+string(x)+' is '+string(s)+'')
```

**Console Window 2. 1**



```
input the number of data point 4

input x data[0;1;2;3]

input y data[1;2;1;10]

input interpolating value 4


   0.   1.    1.  -2.    12.
   1.   2.   -1.   10.   0.
   2.   1.    9.   0.    0.
   3.   10.   0.   0.    0.

 value of y at x=4 is 41
```

## 2. 3 Newton's Gregory Backward Interpolation Method

Let for $y = f(x)$ a given data point is $(y_0, y_1, y_2 \dots . y_n)$, $(x_0, x_1, x_2 \dots . x_n)$ . In this method we estimate $f(x_i)$ or $y_i$ where $x_i \in (x_0, x_n)$. Data point of the independent variable must be equally spaced ($|x_i - x_{i-1}| = constant$). So $x_n = x_0 + nh$ , $n = 0,1,2,3, \dots$ and $h$ is some constant. Finding $y = f(x)$ for $x \in (x_0, x_n)$. Let $x = x_n + uh$ or $u = \frac{x-x_n}{h}$ (2.10)

$$E^u f(x) = f(x + uh)$$

$$E^u f(x_n) = f((x_n + uh)$$

Using (2.10)

$$f(x) = E^u y_n$$

$$f(x)=(E^{-1})^{-u} y_n$$

Using (2.5)

$$f(x) = (1 - \nabla)^{-u} y_n \qquad (2.11)$$

Binomial expansion of $(1 - \nabla)^{-u}$

$$(1 - \nabla)^{-u} = 1 + \frac{u\nabla}{1!} + \frac{u(u+1)\nabla^2}{2!} + \frac{u(u+1)(u+2)\nabla^3}{3!} + \cdots \qquad (2.12)$$

Using (2.11) and (2.12)

$$\boxed{f(x) = y_n + \frac{u\nabla y_n}{1!} + \frac{u(u+1)\nabla^2 y_n}{2!} + \frac{u(u+1)(u+2)\nabla^3 y_n}{3!} + \cdots} \qquad (2.13)$$

Our equation (2.13) is Newton's Backward interpolation formula.

Where $u = \frac{x-x_n}{h}$.

**Formation of Backward difference table using $\nabla y_n = y_n - y_{n-1}$**

Starts the calculations from the last data point.

| $x$ | $y$ | $\nabla y$ | $\nabla^2 y$ | $\nabla^3 y$ |
|---|---|---|---|---|
| $x_0$ | $y_0$ | | | |
| $x_1$ | $y_1$ | $\nabla y_1 = y_1 - y_0$ | | |
| $x_2$ | $y_2$ | $\nabla y_2 = y_2 - y_1$ | $\nabla^2 y_2 = \nabla y_2 - \nabla y_1$ | |
| $x_3$ | $y_3$ | $\nabla y_3 = y_3 - y_2$ | $\nabla^2 y_3 = \nabla y_3 - \nabla y_2$ | $\nabla^3 y_3 = \nabla^2 y_3 - \nabla^2 y_2$ |

**Table 2.3**

**Example 2.2** Using Newton's Backward Interpolation formula calculate the population of the city in year 1925 from given data.

| Year ($x$) | 1891 | 1901 | 1911 | 1921 | 1931 |
|---|---|---|---|---|---|
| Population of city in thousands ($y$) | 46 | 66 | 81 | 93 | 101 |

**Table 2.5**

Backward difference table from Table 2.5

| $x$ | $y$ | $\nabla y$ | $\nabla^2 y$ | $\nabla^3 y$ | $\nabla^4 y$ |
|---|---|---|---|---|---|
| 1891 | 46 | | | | |
| 1901 | 66 | $\nabla y_1 = 20$ | | | |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1911 | 81 | $\nabla y_2 = 15$ | $\nabla^2 y_2 = -5$ |  |  |
| 1921 | 93 | $\nabla y_3 = 12$ | $\nabla^2 y_3 = -3$ | $\nabla^3 y_3 = 2$ |  |
| 1931 | 101 | $\nabla y_4 = 8$ | $\nabla^2 y_4 = -4$ | $\nabla^3 y_4 = -1$ | $\nabla^4 y_4 = -3$ |

**Table 2.4**

Using Table 2.4 and equation (2.13)

$$f(x) = 1 + \frac{8u}{1!} - \frac{4u(u+1)}{2!} - \frac{u(u+1)(u+2)}{3!} - \frac{3u(u+1)(u+2)(u+3)}{4!}$$

Here $h = 10$ and $x_n = 1931$

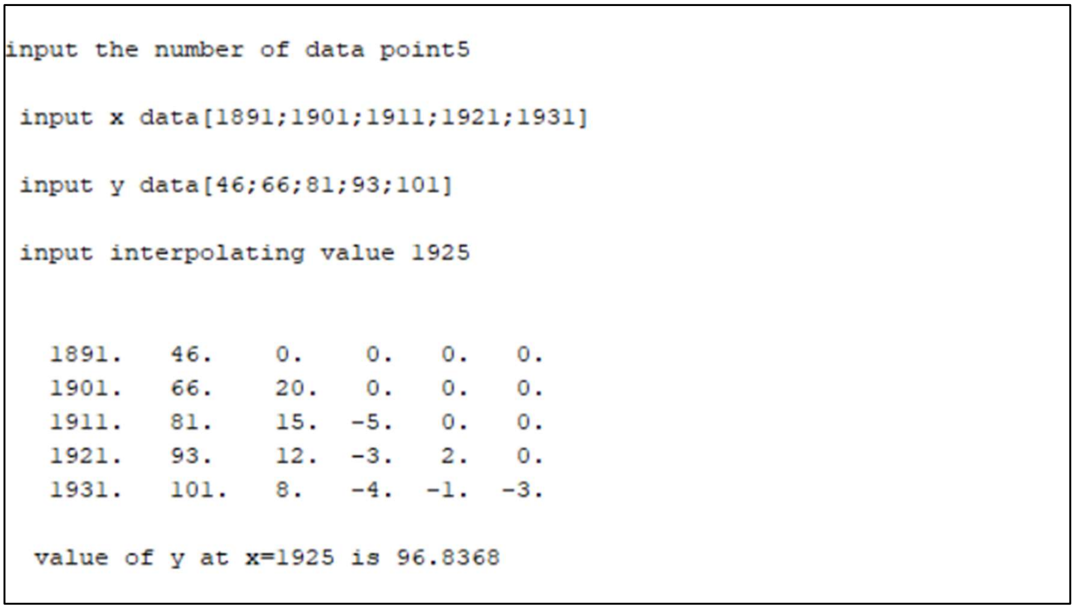
Using (2.10)

$u = \frac{x-193}{10}$,$f(1925) = 96.8368$

**Scilab Code 2. 2**

```
//NEWTON'S Backward INTERPOLATION
clc
clear
n=input('input the number of data point')
//matrix u store table 2.3
u=zeros(n,n+1)
u(:,1)=input(' input x data')
u(:,2)=input(' input y data')
h=abs(u(2,1)-u(1,1))//step size
x=input(' input interpolating value ')

  a=(x-u(n,1))/h

for j=3:n+1//Formation of Backward difference table
  for i=j-1:n
    u(i,j)=u(i,j-1)-u(i-1,j-1)
end
end
disp(u)
//s variable is for summation of terms in equation (2.13)
s=u(n,2)
t=1
  for i=0:n-2
    t=(a+i)*t
    s=s+t*u(n,i+3)/factorial(i+1)
  end
disp(' value of y at x='+string(x)+' is '+string(s)+'')
```

**Console Window 2.2**



```
input the number of data point5

 input x data[1891;1901;1911;1921;1931]

 input y data[46;66;81;93;101]

 input interpolating value 1925


   1891.   46.    0.     0.    0.    0.
   1901.   66.    20.    0.    0.    0.
   1911.   81.    15.   -5.    0.    0.
   1921.   93.    12.   -3.    2.    0.
   1931.   101.   8.    -4.   -1.   -3.

  value of y at x=1925 is 96.8368
```

## 2.4 Lagrange's Interpolation Method

Consider a set of $k+1$ data points of the unequal interval. ($|x_j - x_{j-1}| = \textit{not constant}$)

$$(x_0, y_0), \ldots . (x_j, y_j) \ldots . (x_k, y_k)$$

Where no two $x_j$ are the same. The interpolating function is given by

$$f(x) = \sum_{j=0}^{k} y_j l_j\,(x)$$

Where

$$l_j(x) = \prod_{\substack{0 \le i \le k \\ i \ne j}} \frac{x - x_i}{x_j - x_i}$$

Let's take 3 data $(x_0, y_0), (x_1, y_1), (x_2, y_2)$

$$f(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} y_2$$

### Inverse Lagrange's Interpolation method

This method is used to calculate $x$ for given $f(x)$ where $x \,\epsilon\, (x_0, x_k)$

$$x = \sum_{j=0}^{k} x_j m_j\,(y)$$

Where

$$m_j(y) = \prod_{\substack{0\le i\le k \\ i\ne j}} \frac{y - y_i}{y_j - y_i}$$

**Example 2.3** Compute $f(0.3)$ for the data

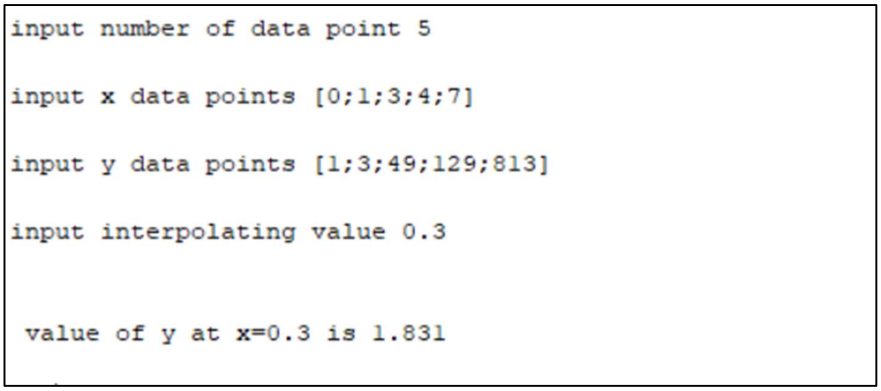
| $x$ | 0 | 1 | 3 | 4 | 7 |
|---|---|---|---|---|---|
| $f$ | 1 | 3 | 49 | 129 | 813 |

by Lagrange's interpolation formula using Scilab.

**Scilab Code 2.3**

```
//Lagrange's Interpolation Method
clc
clear
k=input(' input number of data point ')
x=input(' input x data points ')
y=input(' input y data points ')
a=input(' input interpolating value ')
s=0
for j=1:k
  l(j)=1
  for i=1:k
    if j~=i then
      l(j)=((a-x(i))/(x(j)-x(i)))*l(j)
    end
  end
  s=s+y(j)*l(j)
end
disp(' value of y at x='+string(a)+' is '+string(s)+'')
```

**Console Window 2.3**



```
input number of data point 5

input x data points [0;1;3;4;7]

input y data points [1;3;49;129;813]

input interpolating value 0.3


 value of y at x=0.3 is 1.831
```

# CHAPTER 3
# Curve Fitting

We do experiments and observe data in the form of $(x_i, f(x_i))$ most of the time. Function $f(x)$ is required for further calculation.

Let's consider a simple experiment which is about finding the resistance of ohmic material. Where we are observing voltage for different current. We know that relation between $V$ and $I$ is a straight line and slope corresponds to the resistance of ohmic material. Now we need to find $f(x)$ or a straight for finding resistance.

We can connect all data point $(V_i, I_i)$ to get a straight line. But in reality, connecting all data point does not give a straight line as there are many sources of error which do not lead to a straight line. So, we need to find the approximate straight line and the method by which we find the approximate straight line from given $(x_i, f(x_i))$ data is called Curve Fitting or Line fitting (in this case we are fitting into line curve). The 2nd kind of curve fitting is parabola fitting for a physical problem which is described by the parabolic equation. E.g. Length vs Time period in simple pendulum [ $L = \frac{T^2 g}{4\pi^2}$ ].

## 3.1 Line Fitting

Let we have $(x_i, y_i)$ data points for finding the best fit line $y = ax + b$. To find $a$ and $b$ we need to solve the system of the linear equation such that

$$a\sum x + nb = \sum y$$
$$a\sum x^2 + b\sum x = \sum xy$$

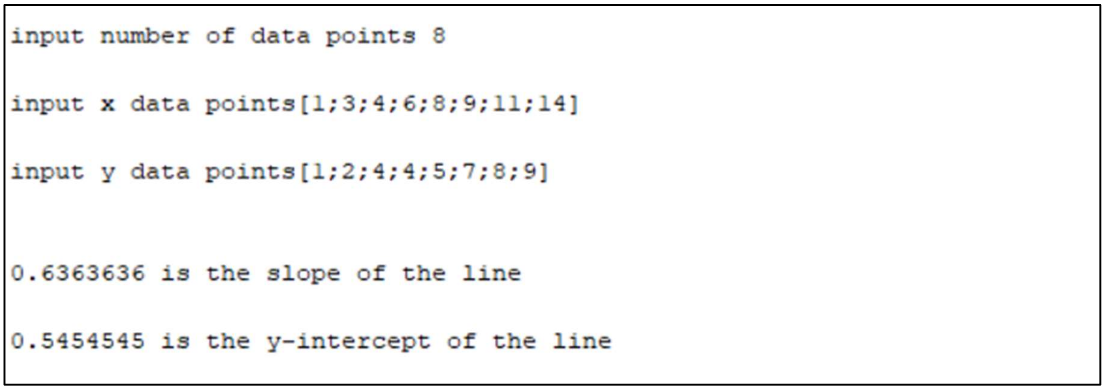
Where $n$ is no. of data points.

**Scilab Code 3.1**

```
//Line Fitting
clc
clear
clf
n=input(' input number of data points ')
x=input(' input x data points')
y=input(' input y data points')
plot(x,y,'*r')
for i=1:n
   z(i)=x(i)*y(i)
```

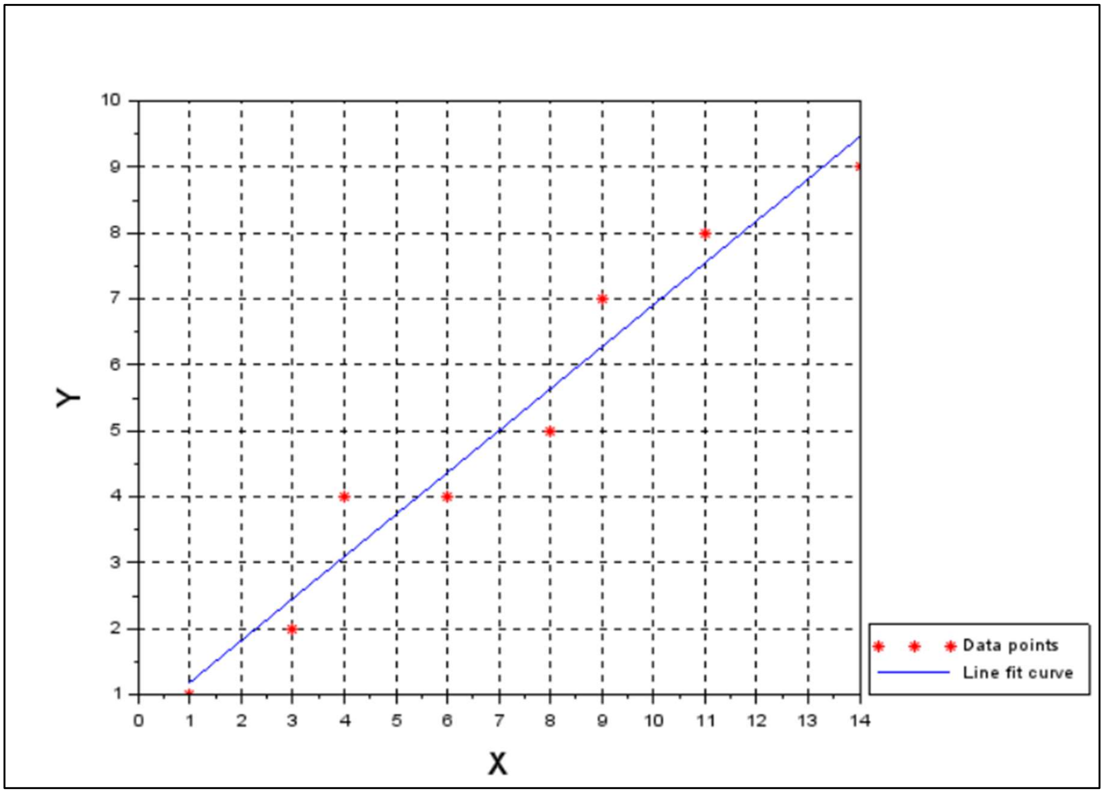
```
end
sx=sum(x)
sy=sum(y)
sz=sum(z)
sx2=sum(x^2)
A=[sx n;sx2 sx]
B=[sy;sz]
C=inv(A)*B
a=C(1,1)//a is the slope of the line
b=C(2,1)//b is the y-intercept of the line
disp(''+string(a)+' is the slope of the line')
disp(''+string(b)+' is the y-intercept of the line')
for i=1:n
   t(i)=a*x(i)+b
end
plot(x,t,'m')
xlabel('X','Fontsize',4)
ylabel('Y','Fontsize',4)
legend('Data points','Line fit curve',-4)
xgrid(3)
```

**Console Window 3. 1**



```
input number of data points 8

input x data points[1;3;4;6;8;9;11;14]

input y data points[1;2;4;4;5;7;8;9]


0.6363636 is the slope of the line

0.5454545 is the y-intercept of the line
```

**Graphic Window 3.1**



## 3.2 Parabola Fitting

Let we need to find best fit parabola $y = ax^2 + bx + c$ for given data point $(x_i, y_i)$ . To find $a, b$ and $c$ we need to solve the system of the linear equation such that

$$a\sum x^2 + b\sum x + nc = \sum y$$
$$a\sum x^3 + b\sum x^2 + c\sum x = \sum xy$$
$$a\sum x^4 + b\sum x^3 + c\sum x^2 = \sum x^2 y$$

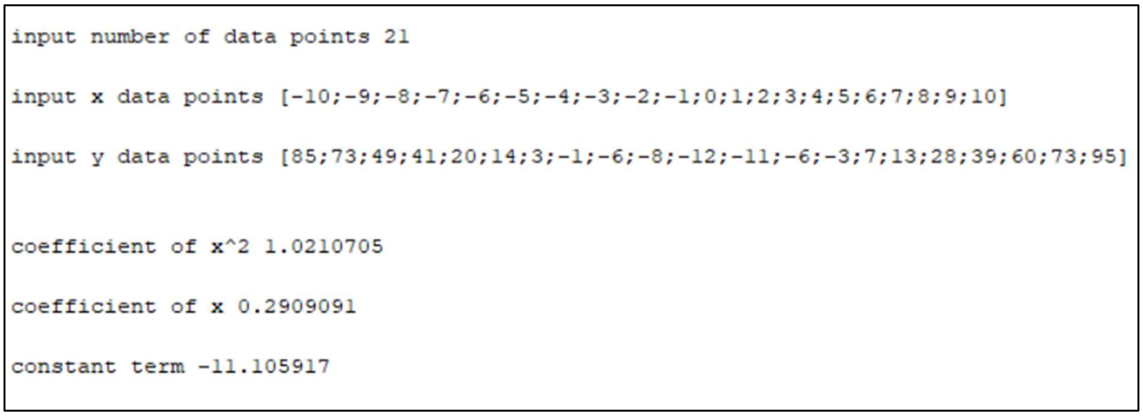
Where $n$ is no. of data points.

**Scilab Code 3.2**

```
//Parabola Fitting
clc
clear
clf
n=input(' input number of data points ')
x=input(' input x data points ')
y=input(' input y data points ')
plot(x,y,'*b')
for i=1:n
  z1(i)=x(i)*y(i)
```

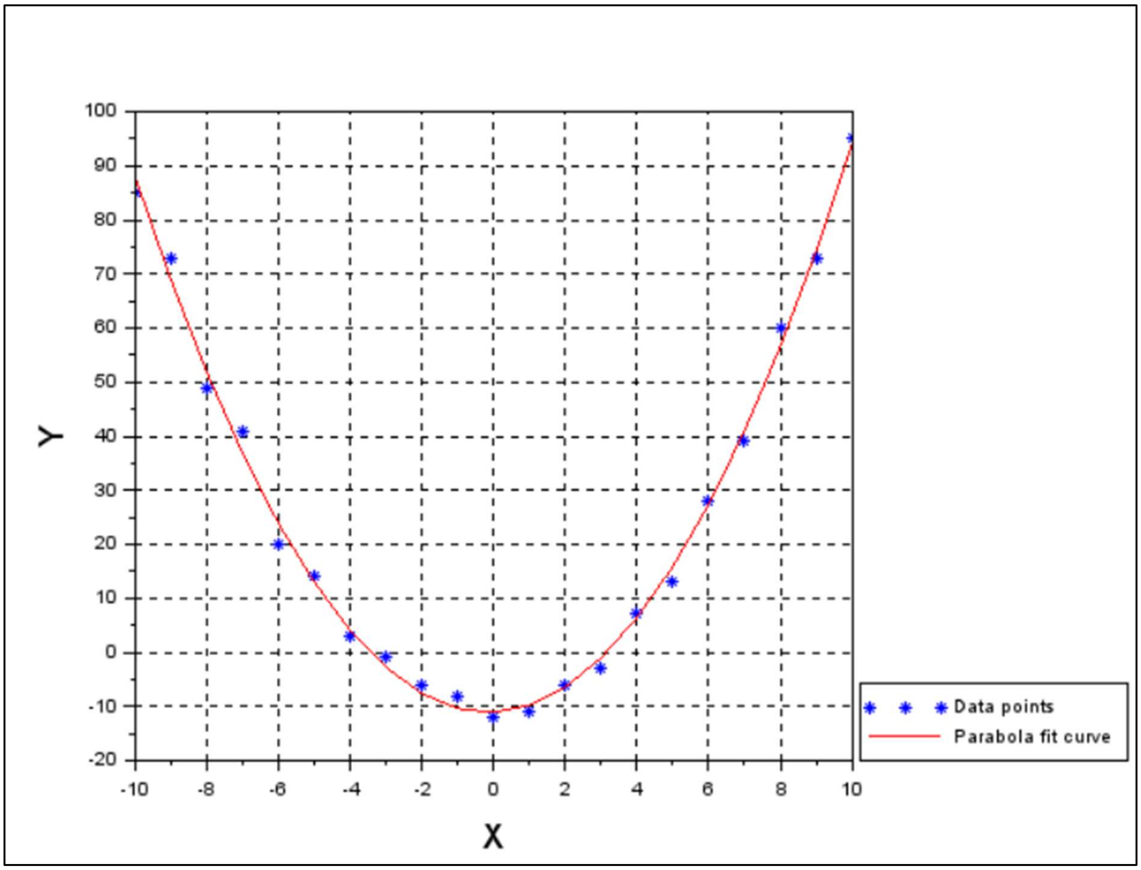
```
   z2(i)=x(i)^2*y(i)
end
sx=sum(x)
sx2=sum(x^2)
sy=sum(y)
sx3=sum(x^3)
sx4=sum(x^4)
sxy=sum(z1)
sx2y=sum(z2)
A=[sx2 sx n;sx3 sx2 sx;sx4 sx3 sx2]
B=[sy;sxy;sx2y]
C=inv(A)*B
a=C(1,1)//y=ax^2+bx+c
b=C(2,1)
c=C(3,1)
disp('coefficient of x^2 '+string(a)+'')
disp('coefficient of x '+string(b)+'')
disp('constant term '+string(c)+'')
for i=1:n
   t(i)=a*x(i)^2+b*x(i)+c
end
plot(x,t,'c')
xlabel('X','Fontsize',4)
ylabel('Y','Fontsize',4)
legend('Data points','Parabola fit curve',-4)
xgrid(3)
```

**Console Window 3. 2**



```
input number of data points 21

input x data points [-10;-9;-8;-7;-6;-5;-4;-3;-2;-1;0;1;2;3;4;5;6;7;8;9;10]

input y data points [85;73;49;41;20;14;3;-1;-6;-8;-12;-11;-6;-3;7;13;28;39;60;73;95]


coefficient of x^2 1.0210705

coefficient of x 0.2909091

constant term -11.105917
```

**Graphic Window 3. 2**



# CHAPTER 4

# Numerical Differentiation

Numerical Differentiation is about finding $y'(x)$ for given $(x_i, y_i)$ data points.

## 4. 1 Equal Interval

When $|x_i - x_{i-1}| = \ constant$, Newton Forward and Newton Backward interpolation formula are used to construct $f(x)$ for given $(x_i, y_i)$ data points. So, we can deduce the Numerical Differentiation formula from Newton Forward and Newton Backward interpolation formula. Instead of constructing $f(x)$ and differentiating it to find $f'(x)$ we will construct a formula for $f'(x)$ from interpolating formula.

**Forward Numerical Differentiation Formula**

From Newton Forward interpolation formula equation (2.9)

$$y(x) = y_0 + \frac{u\Delta y_0}{1!} + \frac{u(u-1)\Delta^2 y_0}{2!} + \frac{u(u-1)(u-2)\Delta^3 y_0}{3!} + \cdots$$

Where $u = \frac{x-x_0}{h}$,$x = x_0 + uh$

Substituting $x$ with $x_0 + uh$ in (2.9)

$$y(x_0 + uh) = y_0 + \frac{u\Delta y_0}{1!} + \frac{(u^2 - u)\Delta^2 y_0}{2!} + \frac{(u^3 - 3u^2 + 2u)\Delta^3 y_0}{3!} + \cdots \qquad (4.1)$$

Differentiating (4.1) with respect to $u$

$$\boxed{y'(x) = \frac{dy}{dx} = \frac{\Delta y_0}{1!\, h} + \frac{(2u-1)\Delta^2 y_0}{2!\, h} + \frac{(3u^2 - 6u + 2)\Delta^3 y_0}{3!\, h} + \cdots \qquad (4.2)}$$

Equation (4.2) is our formula for Numerical Differentiation from the Newton Forward Interpolation formula.

**Backward Numerical Differentiation Formula**

From Newton Backward Interpolation formula equation (2.13)

$$y(x) = y_n + \frac{u\nabla y_n}{1!} + \frac{u(u+1)\nabla^2 y_n}{2!} + \frac{u(u+1)(u+2)\nabla^3 y_n}{3!} + \cdots$$

Where $u = \frac{x-x_n}{h}$,$x = x_n + uh$

Substituting $x$ with $x_n + uh$ in (2.13)

$$y(x_n + uh) = y_n + \frac{u\nabla y_n}{1!} + \frac{(u^2 + u)\nabla^2 y_n}{2!} + \frac{(u^3 + 3u^2 + 2u)\nabla^3 y_n}{3!} + \cdots \qquad (4.3)$$

Differentiating (4.3) with respect to $u$

$$\boxed{y'(x) = \frac{dy}{dx} = \frac{\nabla y_n}{1!\,h} + \frac{(2u + 1)\nabla^2 y_n}{2!\,h} + \frac{(3u^2 + 6u + 2)\nabla^3 y_n}{3!\,h} + \cdots \qquad (4.4)}$$

Equation (4.4) is our formula for Numerical Differentiation from the Newton Backward Interpolation formula.

## 4.2 Unequal Interval

We have seen in chapter 2 Lagrange's interpolation method is used to construct $f(x)$ for an unequal interval ($|x_i - x_{i-1}| = not\ constant$) for given data point $(x_i, y_i)$. In unequal interval data points, we don't derive a separate formula for numerical differentiation. We just differentiate $f(x)$ by constructing it from interpolating formula to get $f'(x)$.

# CHAPTER 5
# Numerical Integration

Numerical Integration is about solving definite integral by discretization of integral limit or finding $\int_{x_0}^{x_n} y\,dx$ for given $(x_i, y_i)$.

Before jumping into Numerical Integration. First, we must have to understand the basic definition of Integration.

We are given to evaluate $I = \int_a^b y\,dx$ , $y = f(x)$. That means we need to calculate area bounded by $y = f(x), y = 0, x = a$ and $x = b$.

Solving integral by slicing bounded area into Rectangular strip (Figure 5.1). Then calculating the individual area of rectangular strip and adding them to get the total area. So, we need to choose the width of the rectangular strip ($h = |x_i - x_{i-1}|$)

Obviously, the length of the rectangle is $f(x_i)$ at $x_i$ (Figure 5.1).

Considering a smaller value of $h$ leads to more accuracy (to make a perfect rectangle).

Number of discrete point in x-axis is $n = \frac{b-a}{h} + 1$.

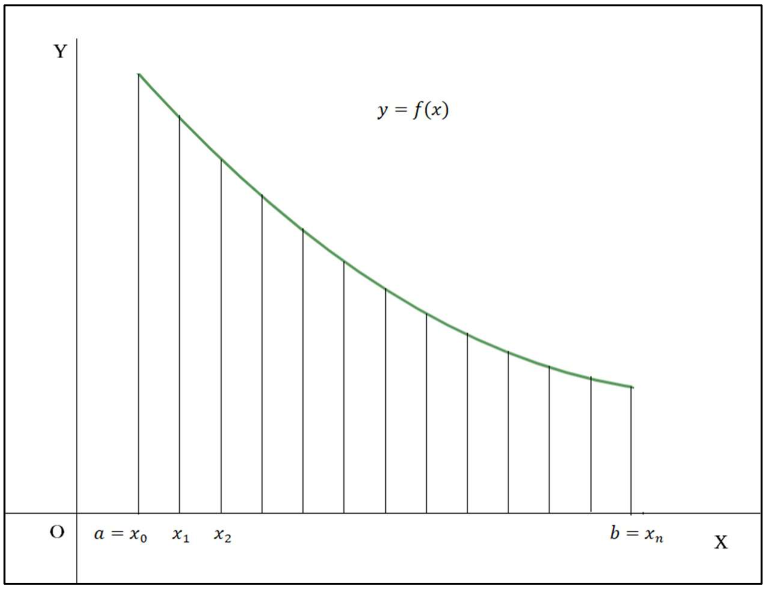


**Figure 5.1**

There are many methods for Numerical Integration but we will focus on the listed below:

(a) Trapezoidal rule

(b) Simpson's 1/3 rule
(c) Simpson's 3/8 rule

All three formulas are deduced from Newton Cotes formula and this formula is derived from the Newton Forward Interpolation formula.

## 5. 1 Newton-Cotes Formula

Let $y = f(x)$ be a function such that $y_0 = f(x_0)\ , y_1 = f(x_1) \dots\dots, y_n = f(x_n)$.

Where $x_n = x_0 + nh$, $h$ is the step size.

From Newton's Forward Interpolation formula equation (2.9)

$$f(x) = y = y_0 + \frac{u\Delta y_0}{1!} + \frac{u(u-1)\Delta^2 y_0}{2!} + \frac{u(u-1)(u-2)\Delta^3 y_0}{3!} + \cdots$$

Where $u = \frac{x-x_0}{h}, x = x_0 + uh$

Now integrate $\int_a^b y\,dx$ where $a = x_0, b = x_n$

$$\int_{x_0}^{x_n} y\,dx = \int_{x_0}^{x_0+nh} y\,dx \tag{5.1}$$

Substituting $x$ with $x_0 + uh$ in (2.9)

$dx = hdu$ and $u = 0$ to $u = n$

From (2.9) and (5.1)

$$\int_{x_0}^{x_0+nh} y\,dx = h\int_0^n y(x_0 + uh)\,du$$

$$= h\int_0^n (y_0 + \frac{u\Delta y_0}{1!} + \frac{(u^2-u)\Delta^2 y_0}{2!} + \frac{(u^3-3u^2+2u)\Delta^3 y_0}{3!} \dots)\,du$$

$$\boxed{\int_{x_0}^{x_0+nh} y\,dx = h[(ny_0 + \frac{n^2\Delta y_0}{2} + \frac{(2n^3-3n^2)\Delta^2 y_0}{12} + \frac{(n^4-4n^3+4n^2)\Delta^3 y_0}{24} \dots)] \ (5.2)}$$

The above equation (5.2) is our Newton-Cotes formula as a parent formula for Numerical Integration methods.

## 5.2 Trapezoidal Rule

Take $n = 1$(this mean only two data point$(x_0, y_0), (x_1, y_1)$). Substitute into newton-cotes formula (5.2)

| $x$ | $y$ | $\Delta y$ |
|---|---|---|
| $x_0$ | $y_0$ | $\Delta y_0 = y_1 - y_0$ |
| $x_1$ | $y_1$ | |

From the above table, we can see that only $\Delta y_0$ is defined. So, terms containing $\Delta^2 y_0, \Delta^3 y_0 \ldots$are redundant.

$$\int_{x_0}^{x_1} y\,dx = h\left(1y_0 + \frac{1}{2}\Delta y_0\right) = h\left[y_0 + \frac{1}{2}(y_1 - y_0)\right] = \frac{h}{2}[y_0 + y_1]$$

Similarly,

$\int_{x_1}^{x_2} y\,dx = \frac{h}{2}[y_1 + y_2], \int_{x_2}^{x_3} y\,dx = \frac{h}{2}[y_2 + y_3] \ldots . \int_{x_{n-1}}^{x_n} y\,dx = \frac{h}{2}[y_{n-1} + y_n]$

$\int_{x_0}^{x_n = x_0 + nh} y\,dx = \int_{x_0}^{x_1} y\,dx + \int_{x_1}^{x_2} y\,dx + \cdots \int_{x_{n-1}}^{x_n} y\,dx$

$$= \frac{h}{2}[y_0 + y_1 + y_1 + y_{2} + y_2 \ldots \ldots . y_{n-1} + y_{n-1} + y_n]$$

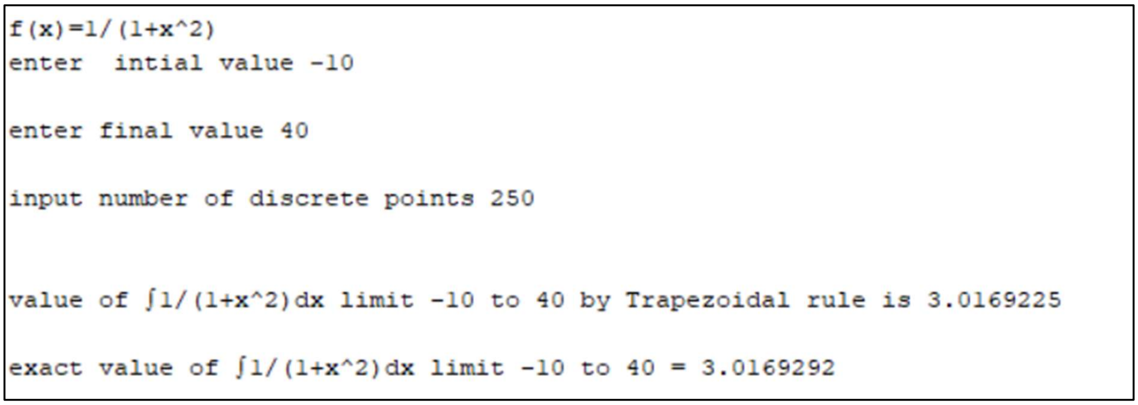
$$\int_{x_0}^{x_n = x_0 + nh} y\,dx = \frac{h}{2}[y_0 + 2(y_1 + y_2 \ldots . + y_{n-1}) + y_n] \qquad (5.3)$$

The above equation (5.3) is the Trapezoidal formula for Numerical Integration. In this formula, any value of $n$ is allowed.

### Scilab Code 5.1

```
//Numerical Integration by Trapezoidal rule
clc
clear
function z=f(x)
   z=1/(1+x^2)
endfunction
disp('f(x)=1/(1+x^2)')
a=input(' enter  intial value ')
b=input(' enter final value ')
n=input(' input number of discrete points ')
h=(b-a)/(n-1)//step size
x(1)=a
y(1)=f(a)
for i=2:n
   x(i)=x(i-1)+h
   y(i)=f(x(i))
end
I=(sum(y)*h)-(h*0.5*(y(1)+y(n)))
disp('value of ∫1/(1+x^2)dx limit '+string(a)+' to '+string(b)+' by Trapezoidal
rule is '+string(I)+'')
z=intg(a,b,f)
disp('exact value of ∫1/(1+x^2)dx limit '+string(a)+' to '+string(b)+' =
'+string(z)+'' )
```

### Console Window 5.1



```
f(x)=1/(1+x^2)
enter  intial value -10

enter final value 40

input number of discrete points 250


value of ∫1/(1+x^2)dx limit -10 to 40 by Trapezoidal rule is 3.0169225

exact value of ∫1/(1+x^2)dx limit -10 to 40 = 3.0169292
```

## 5.3 Simpson's 1/3 rule

Take $n = 2$ and substitute into Newton-Cotes formula $(5.2)$

data points $(x_0, y_0), (x_1, y_1), (x_2, y_2)$

| $\boldsymbol{x}$ | $\boldsymbol{y}$ | $\boldsymbol{\Delta y}$ | $\boldsymbol{\Delta^2 y}$ |
|---|---|---|---|
| $x_0$ | $y_0$ | $\Delta y_0 = y_1 - y_0$ | $\Delta^2 y_0 = \Delta y_1 - \Delta y_0$ |
| $x_1$ | $y_1$ | $\Delta y_1 = y_2 - y_1$ | |
| $x_2$ | $y_2$ | | |

From the above table, we can see that term containing $\Delta^3 y_0, \Delta^4 y_0 \ldots..$ are redundant in newton cote's formula.

$$\int_{x_0}^{x_2} y\,dx = h\left(2y_0 + \frac{4}{2}\Delta y_0 + \frac{1}{12}(16 - 12)\Delta^2 y_0\right)$$

$$= h\left(2y_0 + 2(y_1 - y_0) + \frac{1}{3}\Delta(y_1 - y_0)\right)$$

$$= h\left(2y_0 + 2(y_1 - y_0) + \frac{1}{3}(y_2 - 2y_1 + y_0)\right)$$

$$= \frac{h}{3}(y_0 + 4y_1 + y_2)$$

Similarly,

$\int_{x_2}^{x_4} y\,dx = \frac{h}{3}(y_2 + 4y_3 + y_4)$ ,$\int_{x_4}^{x_6} y\,dx = \frac{h}{3}(y_4 + 4y_5 + y_6)$ ...

$$\int_{x_{n-2}}^{x_n} y\,dx = \frac{h}{3}(y_{n-2} + 4y_{n-1} + y_n)$$

$\int_{x_0}^{x_n = x_0 + nh} y\,dx = \int_{x_0}^{x_2} y\,dx + \int_{x_2}^{x_4} y\,dx + \cdots \int_{x_{n-2}}^{x_n} y\,dx$

$$\boxed{\int_{x_0}^{x_n} y\,dx = \frac{h}{3}(y_0 + 4(y_1 + y_3 + y_5 \ldots y_{n-1}) + 2(y_2 + y_4 + y_6 \ldots y_{n-2}) + y_n)} \quad (5.4)$$

The above equation (5.4) is Simpson's 1/3 rule for Numerical Integration. In this formula, only multiple of 2 is allowed for $n$.

## 5.4 Simpson's 3/8 rule

By substituting $n = 3$ in Newton-Cotes formula (5.2). We get the formula for Simpson’s 3/8 rule

$$\boxed{\int_{x_0}^{x_n} y\,dx = \frac{3h}{8}\left(y_0 + 2(y_3 + y_6 + y_9 \dots y_{n-3}) + 3(y_1 + y_2 + y_4 \dots y_{n-1}) + y_n\right) \quad (5.5)}$$

2nd term in (5.5) containing $y_i$ where $i$ is multiple of 3.

3rd term in (5.5) contains $y_i$ other than $y_0, y_n$ and $y_i$ from 2nd term.

Only multiple of 3 is allowed for $n$ in (5.5)

**Example 5.1** Evaluate $\int_0^6 \frac{dx}{1+x^2}$ by using (i) Trapezoidal rule (ii) Simpson's 1/3 rule (iii) Simpson's 3/8 rule and verify analytically.

Take $n = 6$ , $h = \frac{x_n - x_0}{n} = 1$

| $\boldsymbol{x_i = x_i + ih}$ | $\boldsymbol{y_i = \frac{1}{1+x_i^2}}$ |
|---|---|
| $x_0 = 0$ | $y_0 = 1$ |
| $x_1 = 1$ | $y_1 = 0.5$ |
| $x_2 = 2$ | $y_2 = 0.2$ |
| $x_3 = 3$ | $y_3 = 0.1$ |
| $x_4 = 4$ | $y_4 = 0.0588$ |
| $x_5 = 5$ | $y_5 = 0.0385$ |
| $x_6 = 6$ | $y_6 = 0.027$ |

(a)By Trapezoidal rule

$$\int_0^6 \frac{dx}{1+x^2} = \frac{h}{2}[y_0 + 2(y_1 + y_2 + y_3 + y_4 + y_5) + y_6]$$

$$= 1.4108$$

(b) By Simpson's 1/3 rule

$$\int_0^6 \frac{dx}{1+x^2} = \frac{h}{3}(y_0 + 4(y_1 + y_3 + y_5) + 2(y_2 + y_4) + y_6)$$

$$= 1.3662$$

(c) By Simpson's 3/8 rule

$$\int_0^6 \frac{dx}{1+x^2} = \frac{3h}{8}(y_0 + 2y_3 + 3(y_1 + y_2 + y_4 + y_5) + y_6)$$

$$= 1.3571$$

Solving analytically

$$\int_0^6 \frac{dx}{1+x^2} = [\tan^{-1}(x)]_0^6 = 1.4056$$

**Example 5.2** A river is 80 $m$ wide. The depth $d$ in meters at a distance $x$ from one bank is given by the following table. Find the approximate area of the cross-section.

| $x(m)$ | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|---|---|---|
| $d(m)$ | 0 | 4 | 7 | 9 | 12 | 15 | 14 | 08 | 03 |

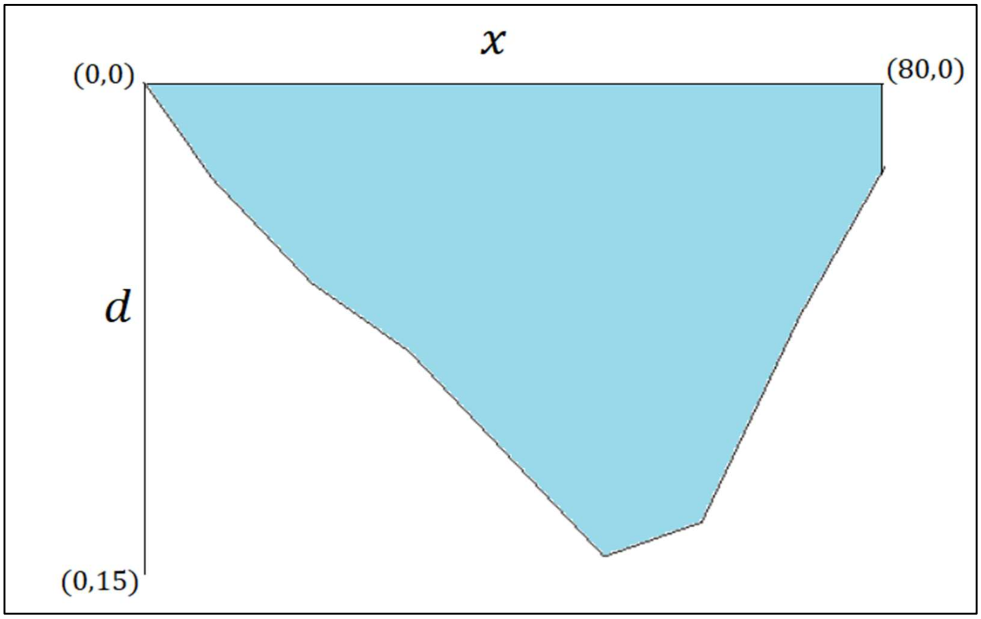


**Figure 5.2**

In this problem cross section area= $\int_0^{80} ydx, h = 10$ and $n = 8$

Let's solve above integral by Simpson's 1/3 rule (5.4).

$$\int_0^{80} ydx = \frac{10}{3}(0 + 4(4 + 9 + 15 + 8) + 2(7 + 12 + 14) + 3)$$

$$=710m^2$$

## 5.5 Monte Carlo method

Solving integral by Monte Carlo method is based on probability.

Let's consider an example $\int_0^2 x^2\,dx$ to see how this method works

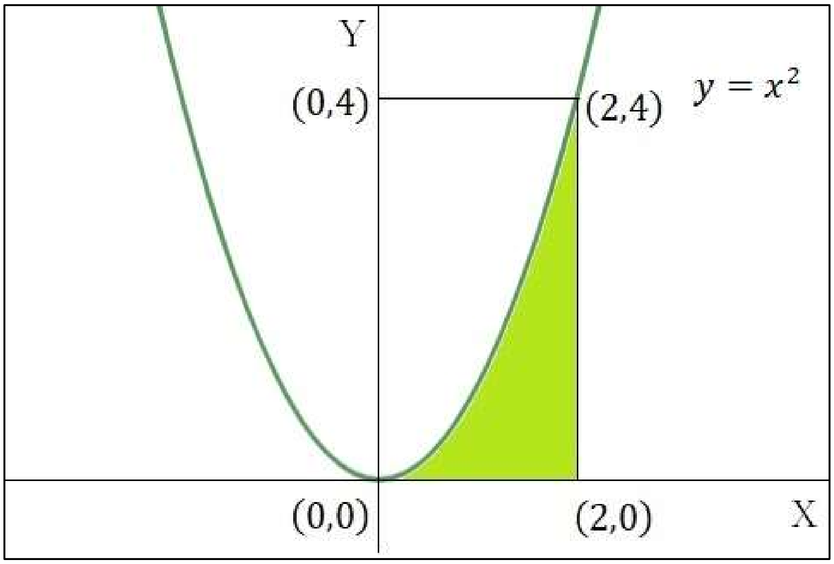


**Figure 5.3**

Solving $\int_0^2 x^2\,dx$ this mean calculating area under curve see Figure 5.3 $((0,0),(2,0),(2,4))$

Steps to implement Monte Carlo method

(a) Consider a Rectangle of size$(2 \times 4)$ Figure 5.3 $((0,0),(2,0),(2,4),(0,4)$.

Pour grain over entire rectangular region $((0,0),(2,0),(2,4),(0,4)$ uniformly.

(b) Count total no. of grains poured and grains inside the region $((0,0),(2,0),(2,4))$

(c) Calculate proportion of the total grains under the curve $(0,0),(2,0),(2,4)) = p = \frac{no.\ of\ grains\ inside\ ((0,0),(2,0),(2,4))}{total\ no.of\ grains.}$

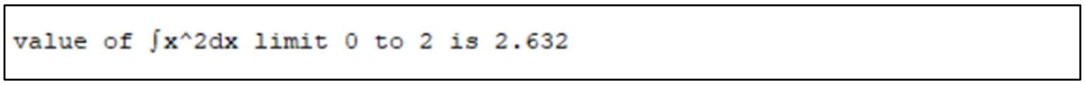
(d)Calculating integral $\int_0^2 x^2\,dx = p \times area\ of\ rectangular\ region$

A higher number of pouring grains leads to more accurate result. Solving integral by the Monte Carlo method manually is not a good idea. So, in computers instead of pouring grains, we generate random numbers and count them.

**Scilab Code 5. 2**

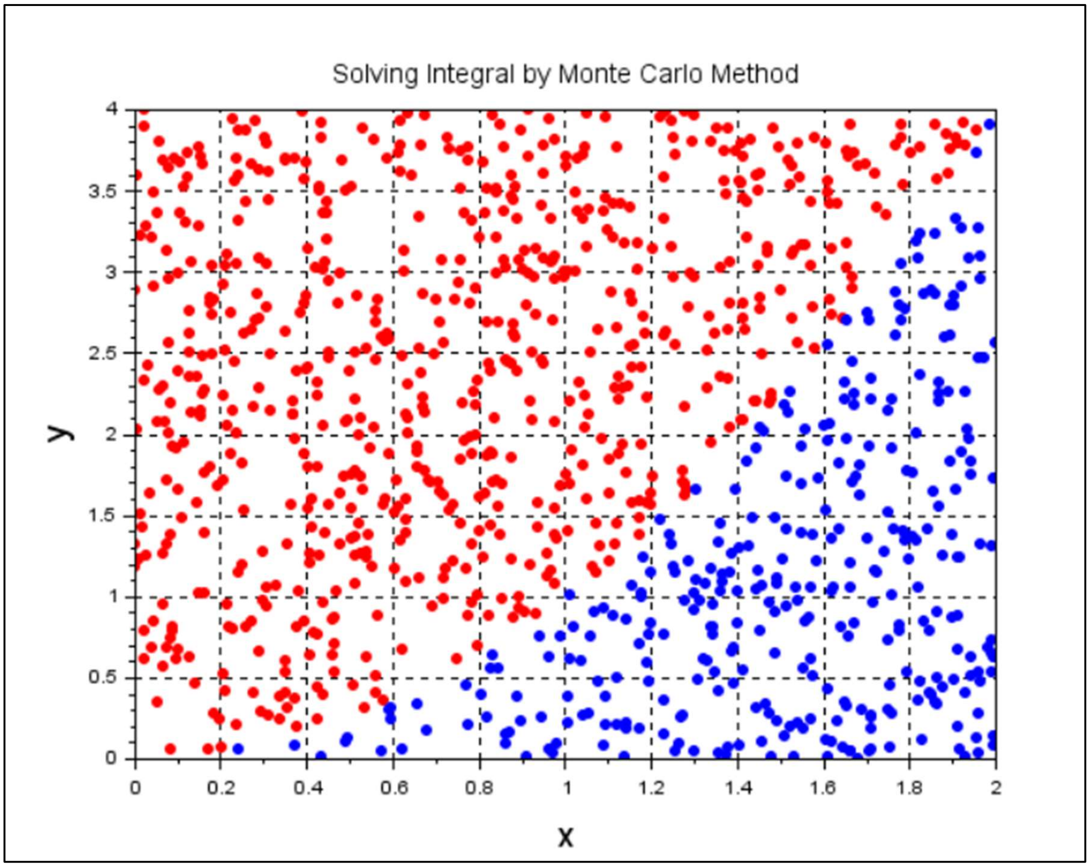
```
//Solving Integral by Monte Carlo Method
clc
clear
clf
inside = 0
n = 1000//total number of random variables
xgrid(4)
xlabel('x','Fontsize',4)
ylabel('y','Fontsize',4)
title('Solving Integral by Monte Carlo Method','Fontsize',3)
for i = 1:n
   x = 2*rand()
   y = 4*rand()
     if y <= x^2 then
        plot(x,y,'g.')
        inside = inside + 1
     else
        plot(x,y,'y.')
     end
    end
 a=(inside/n)*8
disp('value of ∫x^2dx limit 0 to 2 is '+string(a)+'')
```

**Console Window 5. 2**



```
value of ∫x^2dx limit 0 to 2 is 2.632
```

**Graphic Window 5. 2**



# CHAPTER 6

# Ordinary Differential Equation

Many physical problems are expressed in mathematical function and equation involving many variables. These Expressions are called a mathematical model for that physical system, any such mathematical model which involves function and their derivatives are called differential equation.

Many real-world problems are modeled into the differential equation. A few of them are listed below.

(a) Radioactive decay of Radioactive element such as uranium.

(b) Motion of body

(c) population Models

(d) prediction of weather

(e) prediction of stock prices

(f) fluid flow (flow in the river), Motion of airplane, blood flow, swimming of animals, air flow in lungs etc.

(g) Atmospheric pollution

(h) Electric field and potentials

(i) Diffusion of Materials

(k) heat flow and distribution

(l) deformation of the body, elasticity

(m) Modeling cancer growth

(n) pattern formation (e.g. strips in zebra, spots in leopard)

(o) drug delivery to part of the body

(p) propagation of sound wave through medium

(q) Enzyme kinetics

(r) Epidemic models

Versatile use of differential equations motivates us to learn how to solve the differential equation numerically since many differential equations cannot be solved analytically.

## 6. 1 Classification of differential equations

(a) ordinary differential equation (ODE) is an equation that contains one or more derivatives of an unknown function of a single independent variable.

E.g. (a) $\frac{dy}{dx}+\frac{y}{x}=x^2$ (b)$\frac{d^2y}{dx^2}+4y=0$

(b) Partial differential equation (PDE) is an equation involving one or more partial derivates of an unknown function that depends on two or more independent variables.
E.g. (a)$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$ (b) $\frac{\partial u}{\partial t}=c^2\frac{\partial^2 u}{\partial x^2}$

The details of PDE and their numerical methods to solve are discussed in chapter 8.

This chapter is dedicated for numerical methods to solve ODE.

**General Terminology of ODE**

**Order-** Highest order derivative present in ODE

E.g. $x^3\frac{d^3y}{dx^3}-3x^2\frac{d^2y}{dx^2}+6x\frac{dy}{dx}-6y=0$ is of order 3.

**Degree-**Power of highest order derivative in ODE.

E.g. $\left(\frac{d^2y}{ax^2}\right)^2+x^2\left(\frac{dy}{dx}\right)^3=0$ is of order 2 and degree 2.

**Linearity and Homogeneity of $n^{th}$ Order ODE**

An $n^{th}$ order ODE in unknown function $y(x)$ is said to be linear if it can be written in the form

$$y^{(n)}+p_{n-1}(x)y^{(n-1)}+p_{n-2}(x)y^{(n-2)}+\cdots p_0(x)y=r(x)$$

$p_0(x), p_1(x), \ldots p_{n-1}(x)$ are continuous in the defined domain. If $r(x)=0$ for the defined domain then the above equation becomes Homogeneous otherwise it is non-Homogeneous.

## 6. 2 Euler Method

Let's consider 1st order ODE $\frac{dy}{dx} = f(x, y)$ with an initial condition, $y(x_0) = y_0$.

The formula for the Euler method is

$$y_{n+1} = y_n + \Delta x f(x_n, y_n) \tag{6.1}$$

We can calculate $y_i$ from previously calculated $y_{i-1}$ using (6.1). Smaller $\Delta x$ leads to higher accuracy for $y_i$.

**Example 6.1** Find $y(2.2)$ from differential equation $\frac{dy}{dx} = -xy^2$ with the initial condition,$y(2) = 1$.

Here, $f(x, y) = -xy^2$,$x_0 = 2$, $y_0 = 1$

Taking $\Delta x = 0.05$

$y_1 = y_0 + \Delta x f(x_0, y_0)$

$y_1 = y(2.05) = 0.9$

$y_2 = y_1 + \Delta x f(x_1, y_1)$

$y_2 = y(2.1) = 0.817$

$y_3 = y_2 + \Delta x f(x_2, y_2)$

$y_3 = y(2.15) = 0.747$

$y_4 = y_3 + \Delta x f(x_3, y_3)$

$y_4 = y(2.2) = 0.687$

Higher order ODE and System of ODE can be solved using the Euler method. The basic idea behind this is converting ODE into a system of first order ODE and solving them using the Euler method.

**2nd Order ODE**

2nd order ODE $\ddot{x}(t) = f(t, x, \dot{x})$ (6.2)

where $\ddot{x}(t) = \frac{d^2x}{dt^2}, \dot{x}(t) = \frac{dx}{dt}$ with initial condition $x(t_0) = x_0, \dot{x}(t_0) = u_0$

Taking new notation

$\dot{x}(t) = u(t)$ (6.3)

Using 6.2 and 6.3

$\dot{u}(t) = f(t, x, u)$ (6.4)

6.3 and 6.4 is system of 1st order ODE and can be solved using (6.1)

1st data point

$t_1 = t_0 + \Delta t$

$x_1 = x_0 + \Delta t u_0$

$u_1 = u_0 + \Delta t f(t_0, x_0, u_0)$

2nd data point

$t_2 = t_1 + \Delta t$

$x_2 = x_1 + \Delta t u_1$

$u_2 = u_1 + \Delta t f(t_1, x_1, u_1)$

.

.

.

Nth data point

$t_n = t_{n-1} + \Delta t$

$x_n = x_{n-1} + \Delta t u_{n-1}$

$u_n = u_{n-1} + \Delta t f(t_{n-1}, x_{n-1}, u_{n-1})$

**Example 6. 2** Differential equation for the driven damped harmonic oscillator is given below. Describe the position and velocity of a particle by a computer program. $\ddot{x} + 0.5\dot{x} + 3x = 0.4\cos(3t)$ with initial condition $x(0) = 10, u(0) = 0$

Constructing the system of 1st order ODE

$\dot{x}(t) = u(t)$

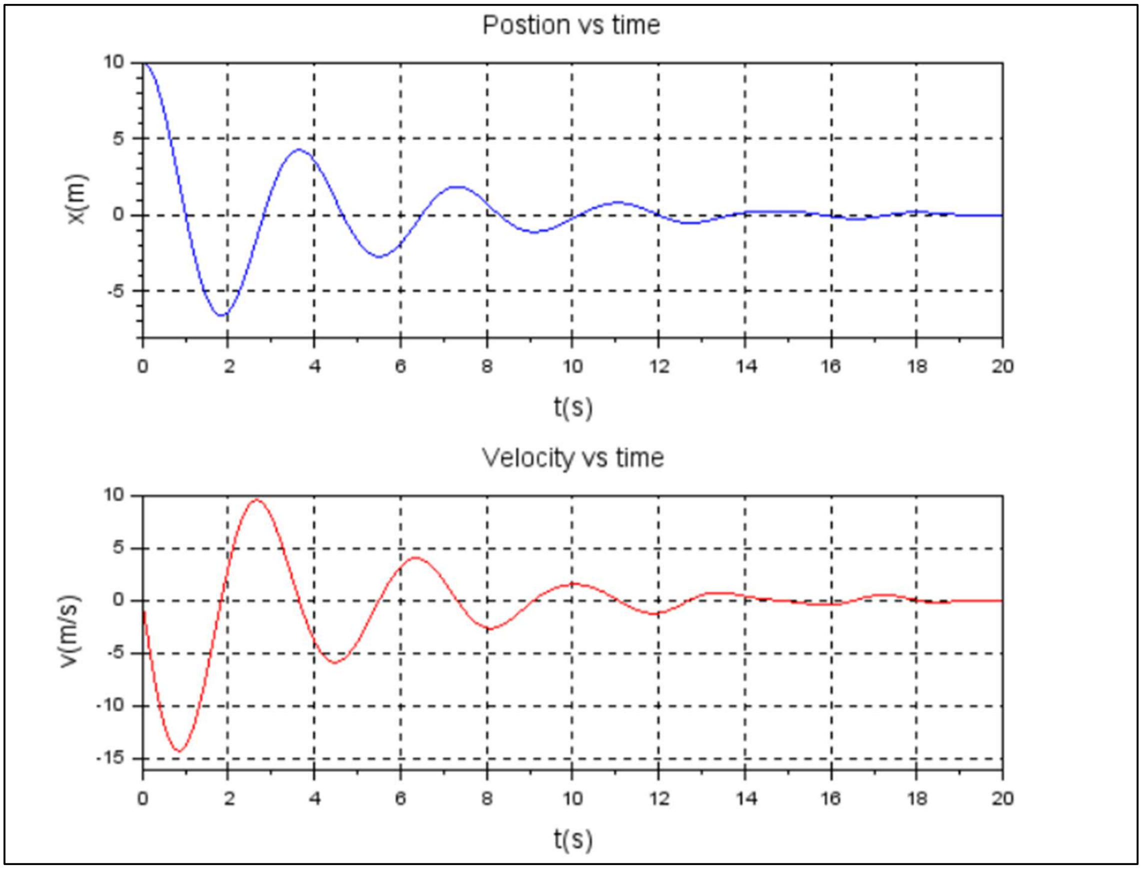
$\dot{u}(t) = 0.4\cos(3t) - 3x - 0.5u$

Taking $\Delta t = 0.01$

**Scilab Code 6. 1**

```
//Example 6.2
clc
clear
clf
function z=f(t, x, u)
  z=0.4*cos(3*t)-0.5*u-3*x
endfunction
t=[0:0.01:20]'//time(s)
x(1)=10//intial postion (m)
u(1)=0//intial velocity(m/s)
for i=1:2000//solving system of 1st order ODE
   x(i+1)=x(i)+0.01*u(i)
   u(i+1)=u(i)+0.01*f(t(i),x(i),u(i))
end
subplot(2,1,1)
plot(t,x,'b')
xgrid(0)
xlabel('t(s)','Fontsize',3)
ylabel('x(m)','Fontsize',3)
title('Postion vs time ','Fontsize',3)
subplot(2,1,2)
plot(t,u,'r')
xgrid(0)
xlabel('t(s)','Fontsize',3)
ylabel('v(m/s)','Fontsize',3)
title('Velocity vs time ','Fontsize',3)
```

**Graphic Window 6.1**



**System of 1st order ODE**

System of 1st order ODE $\dot{x}(t) = f_1(t, x, y)$

$$\dot{y}(t) = f_2(t, x, y)$$

with the initial condition, $x(t_0) = x_0 \; and \; y(t_0) = y_0$

Where $\dot{x}(t) = \frac{dx}{dt}$ and $\dot{y}(t) = \frac{dy}{dt}$

1st data point

$t_1 = t_0 + \Delta t$

$x_1 = x_0 + \Delta t f_1(t_0, x_0, y_0)$

$y_1 = y_0 + \Delta t f_2(t_0, x_0, y_0)$

2nd data point

$t_2 = t_1 + \Delta t$

$x_2 = x_1 + \Delta t f_1(t_1, x_1, y_1)$

$y_2 = y_1 + \Delta t f_2(t_1, x_1, y_1)$

.

.

.

N^th data point

$t_n = t_{n-1} + \Delta t$

$x_n = x_{n-1} + \Delta t f_1(t_{n-1}, x_{n-1}, y_{n-1})$

$y_n = y_{n-1} + \Delta t f_2(t_{n-1}, x_{n-1}, y_{n-1})$

**Example 6. 3** Solve the given initial value problem using the Euler method in Scilab.

$$\frac{dx}{dt} = 3x - 5y$$

$$\frac{dy}{dt} = x - y$$

$$x(0) = 3, y(0) = 1$$

Taking $\Delta t = 0.001$

$x_0 = 3, y_0 = 1, t_0 = 0$

$i = 0,1,2,3 \ldots$

$t_{i+1} = t_i + 0.001$

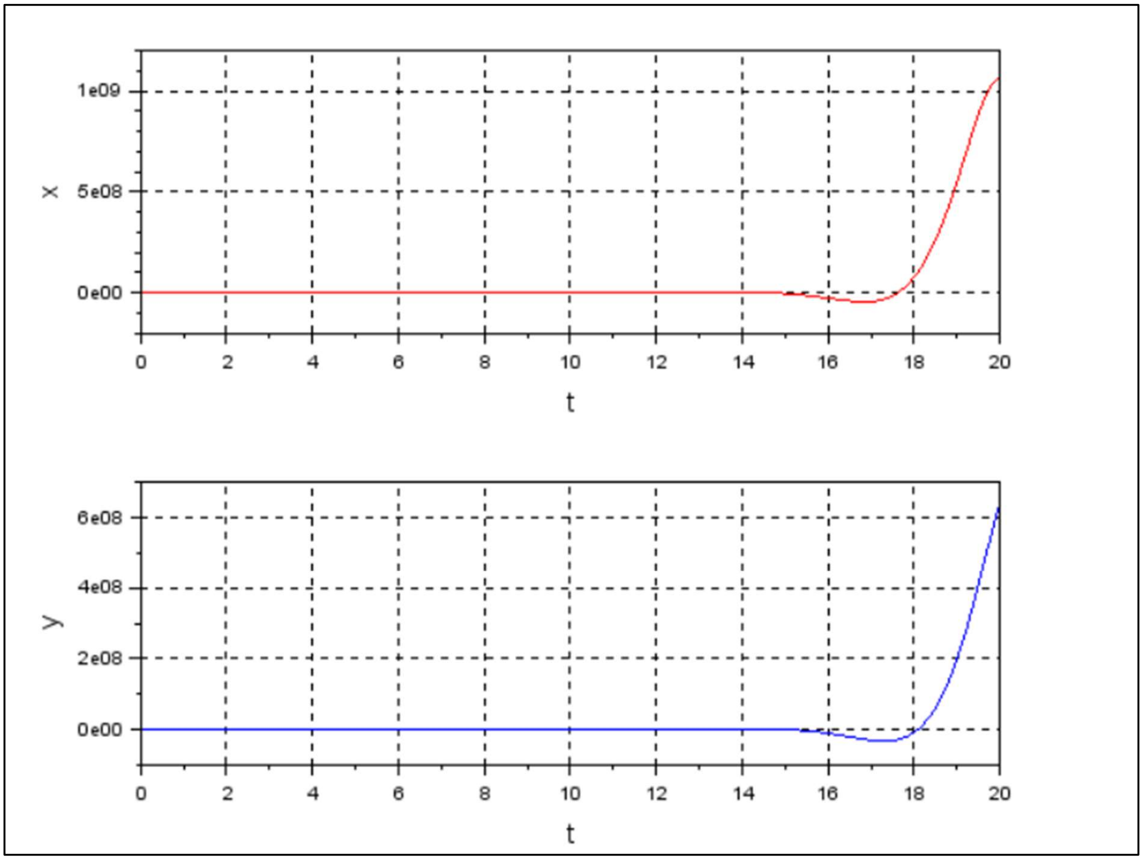
$x_{i+1} = x_i + 0.001 \cdot (3x_i - 5y_i)$

$y_{i+1} = y_i + 0.001 \cdot (x_i - y_i)$

**Scilab Code 6. 2**

```
//Example 6.3
clc
clear
clf
t=[0:0.001:20]'//t
//intial value
x(1)=3//x(0)=3
y(1)=1//y(0)=1
for i=1:20000//solving system of 1st order ODE
   x(i+1)=x(i)+0.001*(3*x(i)-5*y(i))
   y(i+1)=y(i)+0.001*(x(i)-y(i))
end
subplot(2,1,1)
plot(t,x,'r')
xgrid(0)
xlabel('t','Fontsize',3)
ylabel('x','Fontsize',3)
subplot(2,1,2)
plot(t,y,'b')
xgrid(0)
xlabel('t','Fontsize',3)
ylabel('y','Fontsize',3)
```

**Graphic Window 6. 2**



### Higher order ODE

Consider the 3rd order ODE

$y'''(t) = f(t, y, y', y'')$ (6.5)

with initial condition $y(t_0) = y_0, y'(t_0) = u_0, y''(t_0) = a_0$

Where $y'''(t) = \frac{d^3y}{dt^3}, y''(t) = \frac{d^2y}{dt^2}, y'(t) = \frac{dy}{dt}$

Taking new notation

$y'(t) = u(t)$ (6.6)

$y''(t) = u'(t) = a(t)$ (6.7)

Using 6.5, 6.6 and 6.7

$a'(t) = f(t, y, u, a)$ (6.8)

(6.6), (6.7) and (6.8) is system of 1st order ODE and can be solved using (6.1)

1st data point

$t_1 = t_0 + \Delta t$

$y_1 = y_0 + \Delta t u_0$

$u_1 = u_0 + \Delta t a_0$

$a_1 = a_0 + \Delta t f(t_0, y_0, u_0, a_0)$

2nd data point

$t_2 = t_1 + \Delta t$

$y_2 = y_1 + \Delta t u_1$

$u_2 = u_1 + \Delta t a_1$

$a_2 = a_1 + \Delta t f(t_1, y_1, u_1, a_1).$

.

.

.

Nth data point

$t_n = t_{n-1} + \Delta t$

$y_n = y_{n-1} + \Delta t u_{n-1}$

$u_n = u_{n-1} + \Delta t a_{n-1}$

$a_n = a_{n-1} + \Delta t f(t_{n-1}, y_{n-1}, u_{n-1}, a_{n-1})$

We can extend the above notion to solve higher order ODE.

**Example 6.4** Solve the given initial value problem using the Euler method in Scilab.

$y''' - y'' + 100y' - 100y = 0 \, , y(0) = 4, y'(0) = 11, y''(0) = -299$

Constructing the system of 1st order ODE

$y'(t) = u(t)$

$u'(t) = a(t)$

$a'(t) = a - 100u + 100y$

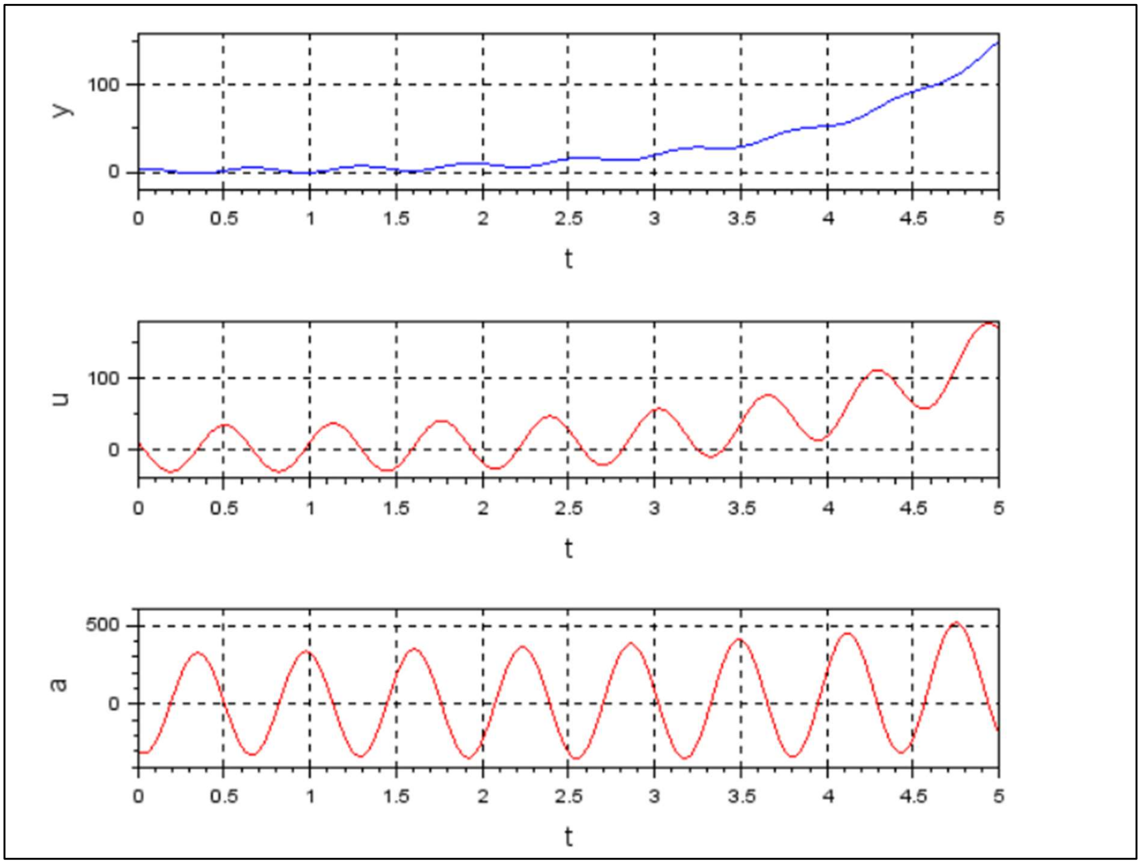
$y(0) = 4, u(0) = 11, a(0) = -299$

Taking $\Delta t = 0.001$

**Scilab code 6. 3**

```
//Example 6.4
clc
clear
clf
t=[0:0.001:5]'//t
//intial value
y(1)=4//y(0)=4
u(1)=11//u(0)=1
a(1)=-299// a(0)=-299
for i=1:5000//solving system of 1st order ODE
   y(i+1)=y(i)+0.001*u(i)
   u(i+1)=u(i)+0.001*a(i)
   a(i+1)=a(i)+0.001*(a(i)-100*u(i)+100*y(i))
end
subplot(3,1,1)
plot(t,y,'b')
xgrid(0)
xlabel('t','Fontsize',3)
ylabel('y','Fontsize',3)
subplot(3,1,2)
plot(t,u,'r')
xgrid(0)
xlabel('t','Fontsize',3)
ylabel('u','Fontsize',3)
subplot(3,1,3)
plot(t,a,'r')
xgrid(0)
xlabel('t','Fontsize',3)
ylabel('a','Fontsize',3)
```

## Graphic Window 6. 3



**System of Higher order ODE**

Let's consider the system of second order ode

$\ddot{x}(t) = f_1(t, x, \dot{x}, y, \dot{y})$

$\ddot{y}(t) = f_2(t, x, \dot{x}, y, \dot{y})$

Where $\ddot{x}(t) = \frac{d^2x}{dt^2}, \ddot{y}(t) = \frac{d^2y}{dt^2}$ $, \dot{x}(t) = \frac{dx}{dt}, \dot{y}(t) = \frac{dy}{dt}$

With initial condition $x(t_0) = x_0, \dot{x}(t) = u_0, y(t_0) = y_0, \dot{y}(t) = v_0$

Considering new variable $u$ and $v$ to make system of first order ode

$\dot{x}(t) = u(t)$

$\dot{y}(t) = v(t)$

$\dot{u}(t) = f_1(t, x, u, y, v)$

$\dot{v}(t) = f_2(t, x, u, y, v)$

Applying Euler method in above system of first ODE

1st data point

$t_1 = t_0 + \Delta t$

$x_1 = x_0 + \Delta t u_0$

$y_1 = y_0 + \Delta t v_0$

$u_1 = u_0 + \Delta t f_1(t_0, x_0, u_0, y_0, v_0)$

$v_1 = v_0 + \Delta t f_2(t_0, x_0, u_0, y_0, v_0)$

2nd data point

$t_2 = t_1 + \Delta t$

$x_2 = x_1 + \Delta t u_1$

$y_2 = y_1 + \Delta t v_1$

$u_2 = u_1 + \Delta t f_1(t_1, x_1, u_1, y_1, v_1)$

$v_2 = v_1 + \Delta t f_2(t_1, x_1, u_1, y_1, v_1)$

.

.

.

N^th data point

$t_n = t_{n-1} + \Delta t$

$x_n = x_{n-1} + \Delta t u_{n-1}$

$y_n = y_{n-1} + \Delta t v_{n-1}$

$u_n = u_{n-1} + \Delta t f_1(t_{n-1}, x_{n-1}, u_{n-1}, y_{n-1}, v_{n-1})$

$v_n = v_{n-1} + \Delta t f_2(t_{n-1}, x_{n-1}, u_{n-1}, y_{n-1}, v_{n-1})$

We can extend the above notion to solve the system of higher order ODE.

**Example 6. 5** Equation of 2 mass coupled oscillator is given below with initial condition, describe the position and velocity of both masses using Scilab.

$\ddot{x}(t) = -6x + 2y$ , $\ddot{y}(t) = 2x - 6y$ , $x(0) = 1, \dot{x}(0) = 0, y(0) = 0, \dot{y}(0) = 0$

Constructing system of 1st order ODE

$\dot{x}(t) = u(t)$

$\dot{y}(t) = v(t)$

$\dot{u}(t) = -6x + 2y$

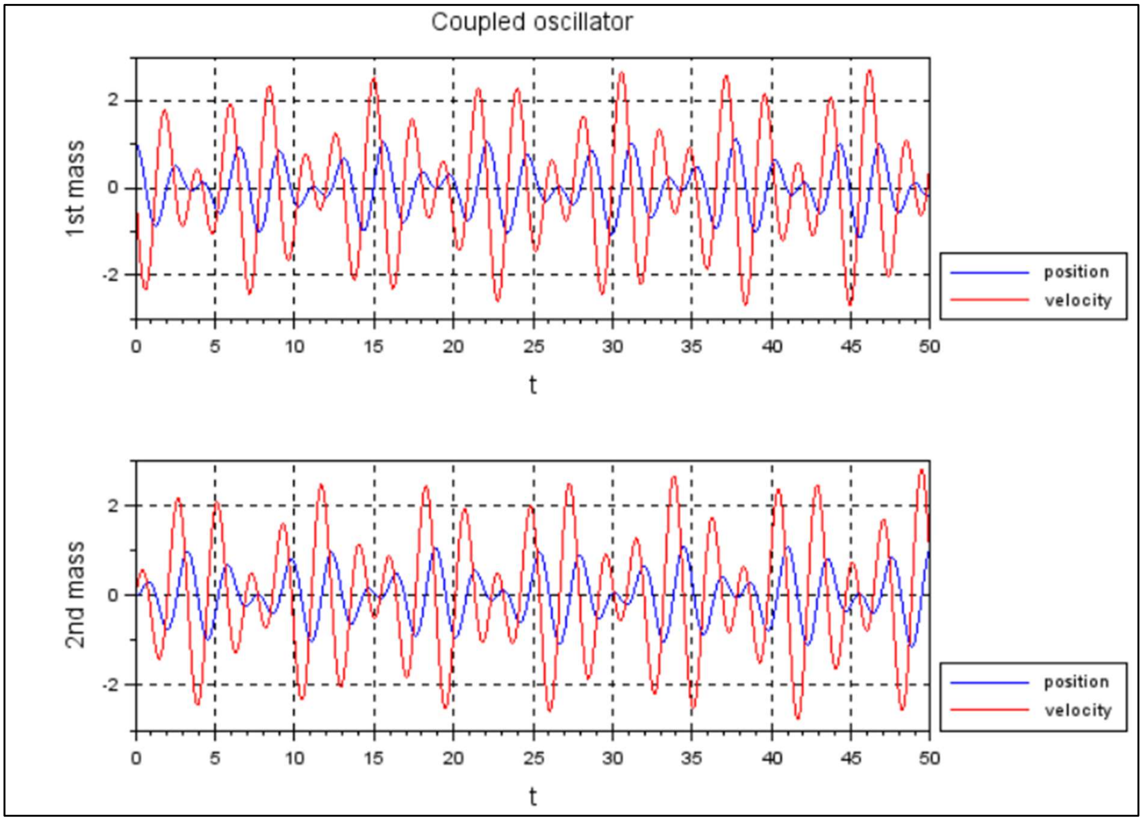
$\dot{v}(t) = 2x - 6y$

$x(0) = 1, u(0) = 0, y(0) = 0, v(0) = 0$ , Taking $\Delta t = 0.001$

**Scilab code 6. 4**

```
//Example 6.5
clc
clear
clf
t=[0:0.001:50]'//time(s)
x(1)=1//intial postion(m) of first mass
y(1)=0//intial postion(m) of second mass
u(1)=0//intial velocity(m/s) of first mass
v(1)=0//intial velocity(m/s) of second mass
for i=1:50000//solving system of 1st order ODE
   x(i+1)=x(i)+0.001*u(i)
   y(i+1)=y(i)+0.001*v(i)
   u(i+1)=u(i)+0.001*(-6*x(i)+2*y(i))
   v(i+1)=v(i)+0.001*(2*x(i)-6*y(i))
end
subplot(2,1,1)
plot(t,x,'b')
plot(t,u,'c')
xgrid(3)
xlabel('t','Fontsize',3)
ylabel('1st mass','Fontsize',3)
title('Coupled oscillator','Fontsize',3)
legend('position','velocity',-4)
subplot(2,1,2)
plot(t,y,'r')
plot(t,v,'m')
xgrid(3)
xlabel('t','Fontsize',3)
ylabel('2nd mass','Fontsize',3)
legend('position','velocity',-4)
```

## Graphic Window 6. 4



## 6. 3 Runge–Kutta method

### Order 2

Let 1st order ODE $\frac{dy}{dx} = f(x, y), y(x_0) = y_0, h$ is the step size.

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

Where $k_1 = hf(x_n, y_n)$

$$k_2 = hf(x_{n+h}, y_n + k_1)$$

### Order 4

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Where $k_1 = hf(x_n, y_n)$

$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

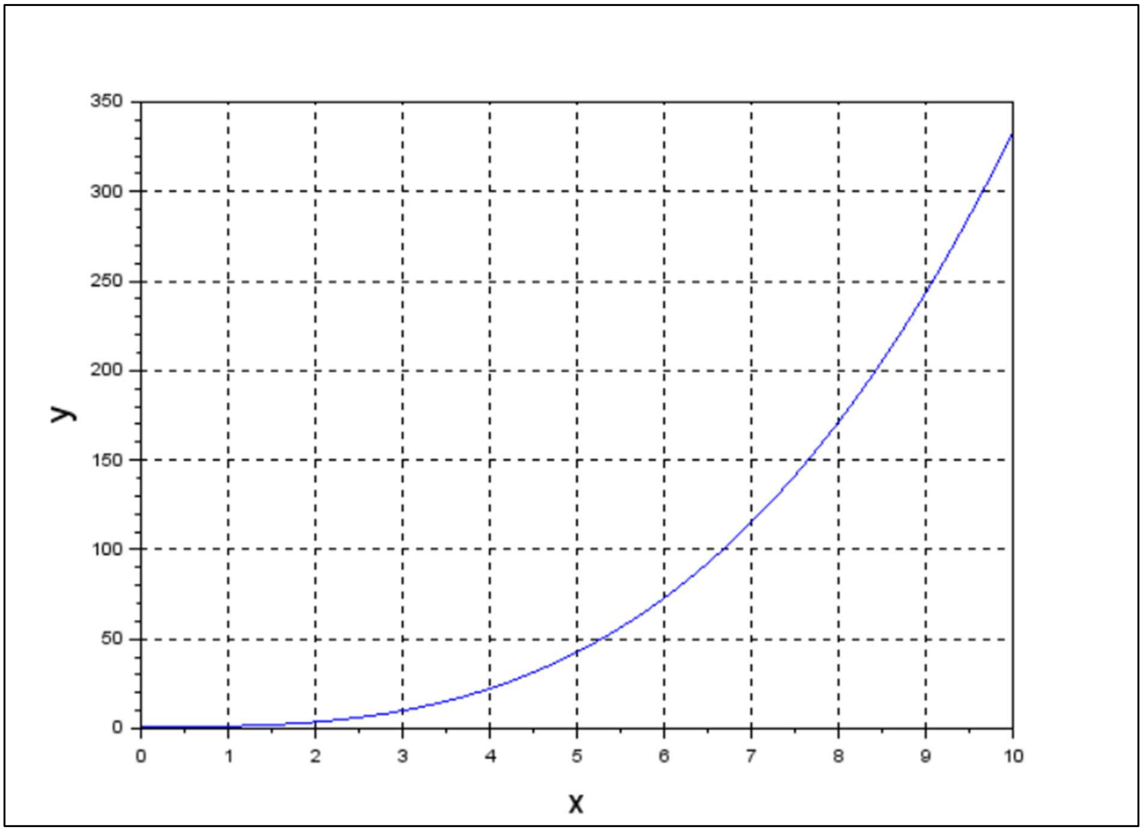
$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

### Scilab code 6. 5

```
//Runge–Kutta Method order 4
clc
clear
clf
function z=f(x, y)
   z=x^2
endfunction
//initial condition
x(1)=0
y(1)=1
h=0.1//step size
for i=1:100
   k1(i)=h*f(x(i),y(i))
   k2(i)=h*f(x(i)+h/2,y(i)+k1(i)/2)
   k3(i)=h*f(x(i)+h/2,y(i)+k2(i)/2)
   k4(i)=h*f(x(i)+h,y(i)+k3(i))
   y(i+1)=y(i)+(k1(i)+2*k2(i)+2*k3(i)+k4(i))/6
   x(i+1)=x(i)+h
end
plot(x,y,'b')
xgrid(3)
xlabel('x','Fontsize',4)
ylabel('y','Fontsize',4)
```

**Graphic Window 6. 5**



# CHAPTER 7
# Linear Algebra

A linear equation in $n$ variables $x_1, x_2, \dots, x_n$ is an equation of the form

$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b.$

### System of Linear Equations

A system of linear equation in $n$ variables $x_1, x_2, \dots, x_n$ is a finite collection of linear equations in these variables. A system of m linear equation in these n variables can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ . \qquad . \qquad\quad &. \\ . \qquad . \qquad\quad &. \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{7.1}$$

A set of values of unknowns $x_1, x_2, \dots, x_n$ which simultaneously satisfy the system of linear equation are called solution. The set of all solutions of a system of linear equation is called the solution set. If $x_1, x_2, \dots, x_n$ are all zero, then the solution is said to be a trivial solution Otherwise, the solution is said to be non-trivial.

A system of linear equations is said to be consistent if it has at least one solution.

A system of linear equations is said to be inconsistent if it has no solution.

**Given any system of linear equations precisely one of the following three is true:**

(a) The system has exactly one solution.

(b) The system has infinitely many solutions.

(c) The system has no solution.

The system of linear equations (7.1) may be written as

$AX = B$

Where $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$, $B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$, $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Augmented matrix of the system of linear equations (7.1)

$$[A|B] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right]$$

## ELEMENTARY ROW TRANSFORMATIONS

The following three operations applied on the rows of matrix are called elementary row operations:

(a) Interchange of any two rows,$R_i \Leftrightarrow R_j$.

(b) Multiplying the elements of a row by a non-zero scalar, $R_i \to kR_i$.

(c) Adding to the elements of a row, the constant times the corresponding elements of another row, $R_i \to R_i + kR_j$.

### Row Echelon Form of a Matrix

A matrix is said to be in row echelon form if it has the following three properties:

1. If a row does not consist entirely of zeros, then the first non-zero entry in the row should be 1, called the leading 1(pivot entry).

2. If any two successive rows that do not consist entirely of zeros, then the leading 1 in the lower row occurs farther to the right than the leading 1 in the upper row, i.e., the leading 1 appears farther to the right as we move down the rows of the matrix.

3. If there are any rows that consist entirely of zeros, then they are grouped together at the bottom of the matrix.

A column containing a leading 1 is called a pivot column, while a column having no leading 1 called a non-pivot column.

**Transforming a Matrix to Row Echelon Form Using Elementary Row Operations**

1. Interchange the top row with another row, if necessary, to bring a non-zero entry at the top of the column.

2.If the entry in the first row and first column $a_{11}$. divide the first row by $a_{11}$ in order to get a leading 1.

3. Add  suitable multiple of the top row to the rows below so that all entries below the leading 1 become zeros.

4. Now cover the top row of the matrix and start again with step 1 Applied to the submatrix. Continue in this way until the entire matrix is in row echelon form.

**Example 7. 1** Reduce the matrix $A = \begin{bmatrix} 0 & 1 & 3 & -4 \\ 2 & 4 & 8 & 6 \\ 2 & -4 & 0 & 1 \end{bmatrix}$ to row echelon form by using elementary row operations.

$$R_1 \Leftrightarrow R_2$$

$$\begin{bmatrix} 2 & 4 & 8 & 6 \\ 0 & 1 & 3 & -4 \\ 2 & -4 & 0 & 1 \end{bmatrix}$$

$$R_1 \to \frac{1}{2} R_1$$

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 1 & 3 & -4 \\ 2 & -4 & 0 & 1 \end{bmatrix}$$

$$R_3 \to R_3 - 2R_1$$

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 1 & 3 & -4 \\ 0 & -8 & -8 & -5 \end{bmatrix}$$

$$R_3 \to R_3 + 8R_2$$

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 1 & 3 & -4 \\ 0 & 0 & 16 & -37 \end{bmatrix}$$

$$R_3 \to \frac{1}{16} R_3$$

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 1 & 3 & -4 \\ 0 & 0 & 1 & -37/16 \end{bmatrix}$$

The last matrix is row echelon form.

**Reduced Row Echelon Form**

A matrix is said to be in reduced row echelon form if it satisfies the following properties:

1. It is in row echelon form.

2. Each column that contains a leading 1 has zeros everywhere else in that column.

**Transforming a Matrix to Reduced Row Echelon Form Using Elementary Row Operations**

To reduce a matrix to reduced row echelon form, we first reduce it to row echelon form, we then apply the additional step:

Beginning with last non-zero row and working upward we add suitable multiples of each row to the rows above to get zeros above the leading 1′s.

**Example $7.2$** Reduce the matrix $A = \begin{bmatrix} 1 & 2 & -1 & 6 \\ 3 & 8 & 9 & 10 \\ 2 & -1 & 2 & -2 \end{bmatrix}$ to reduced row echelon form by using elementary row operations.

$$R_2 \rightarrow R_2 - 3R_1$$

$$R_3 \rightarrow R_3 - 2R_1$$

$$\begin{bmatrix} 1 & 2 & -1 & 6 \\ 0 & 2 & 12 & -8 \\ 0 & -5 & 4 & -14 \end{bmatrix}$$

$$R_2 \rightarrow \frac{1}{2} R_2$$

$$\begin{bmatrix} 1 & 2 & -1 & 6 \\ 0 & 1 & 6 & -4 \\ 0 & -5 & 4 & -14 \end{bmatrix}$$

$$R_3 \rightarrow R_3 + 5R_2$$

$$\begin{bmatrix} 1 & 2 & -1 & 6 \\ 0 & 1 & 6 & -4 \\ 0 & 0 & 34 & -34 \end{bmatrix}$$

$$R_3 \to \frac{1}{34} R_3$$

$$\begin{bmatrix} 1 & 2 & -1 & 6 \\ 0 & 1 & 6 & -4 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

$$R_1 \to R_1 + R_3$$

$$R_2 \to R_2 - 6R_3$$

$$\begin{bmatrix} 1 & 2 & 0 & 5 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

$$R_1 \to R_1 - 2R_2$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

The last matrix is in reduced row echelon form.

There are many methods to get solution of system of linear equation out of which four are listed below.

(a) Gaussian Elimination Method

(b) Gauss-Jordan Elimination Method

(c) Jacobi Method

(d) Gauss-Seidel Method

(a) and (b) are the direct method which gives exact solution of linear system $AX = B$.

(c) and (d) are iterative method which gives approximate solution of a linear system $AX = B$.

## 7. 1 Gaussian Elimination Method

Let's consider $AX = B$, The steps for solving the system of linear equations by Gaussian Elimination Method is given below:

1. Write the augmented matrix $[A|B]$ of the system.

2. Use the elementary row operations to reduce the augmented matrix $[A|B]$ to a matrix $[C|D]$ in row echelon form.

3. Write the linear system corresponding to the row echelon matrix $[C|D]$ and use back substitution to obtain the solution.

Types of Solution

(a) Inconsistent solution – If any row of the form $[0 \quad 0 \quad \cdots \quad 0|c]$ occur at any stage of the Gaussian Elimination method.

(b) Unique solution- when every column of $C$ is pivot column.

(c) Infinitely many solution-when non-pivot column are present in $C$. The variable corresponding to non-pivot column are called independent that can take on any real values, while those corresponding to pivot columns are called dependent variable the values of the dependent variable are then determined in term of independent variable by making use of back substitution.

**Example 7. 3** Solve the following system of linear equations using Gaussian Elimination Method.

$$x + 2y + 3z = 1$$

$$x + 3y + 5z = 2$$

$$2x + 5y + 9z = 3$$

$$[A|B] = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 1 & 3 & 5 & 2 \\ 2 & 5 & 9 & 3 \end{array}\right]$$

Transforming $[A|B]$ into row echelon form

$$R_2 \to R_2 - R_1$$

$$R_3 \to R_3 - 2R_1$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 1 & 3 & 1 \end{array}\right]$$

$$R_3 \to R_3 - R_2$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

linear system corresponding to the above row echelon matrix

$$x + 2y + 3z = 1$$

$$y + 2z = 1$$

$$z = 0$$

Getting values by back substitution

$z = 0\ , y = 1, x = -1.$

**Example 7. 4** Solve the following system of linear equations using Gaussian Elimination Method.

$$3x_1 - 6x_2 + 3x_4 = 9$$

$$-2x_1 + 4x_2 + 2x_3 - x_4 = -11$$

$$4x_1 - 8x_2 + 6x_3 + 7x_4 = -5$$

$$[A|B] = \left[\begin{array}{cccc|c} 3 & -6 & 0 & 3 & 9 \\ -2 & 4 & 2 & -1 & -11 \\ 4 & -8 & 6 & 7 & -5 \end{array}\right]$$

Transforming $[A|B]$ into row echelon form

$$R_1 \rightarrow \frac{1}{3} R_1$$

$$\left[\begin{array}{cccc|c} 1 & -2 & 0 & 1 & 3 \\ -2 & 4 & 2 & -1 & -11 \\ 4 & -8 & 6 & 7 & -5 \end{array}\right]$$

$$R_2 \rightarrow R_2 + 2R_1$$

$$R_3 \rightarrow R_3 - 4R_1$$

$$\left[\begin{array}{cccc|c} 1 & -2 & 0 & 1 & 3 \\ 0 & 0 & 2 & 1 & -5 \\ 0 & 0 & 6 & 3 & -17 \end{array}\right]$$

$$R_2 \rightarrow \frac{1}{2} R_2$$

$$\left[\begin{array}{cccc|c} 1 & -2 & 0 & 1 & 3 \\ 0 & 0 & 1 & 1/2 & -5/2 \\ 0 & 0 & 6 & 3 & -17 \end{array}\right]$$

$$R_3 \rightarrow R_3 - 6R_2$$

$$\left[\begin{array}{cccc|c} 1 & -2 & 0 & 1 & 3 \\ 0 & 0 & 1 & 1/2 & -5/2 \\ 0 & 0 & 0 & 0 & -2 \end{array}\right]$$

The last row of the above row echelon matrix is of the form $[0 \quad 0 \quad \cdots \quad 0|c]$. So, this linear system of equations has no solution.

linear system corresponding to the above row echelon matrix

$$x_1 - 2x_2 + x_4 = 3$$

$$x_3 + \tfrac{1}{2}x_4 = -\tfrac{5}{2}$$

$$0 = -2$$

**Example 7.5** Solve the following system of linear equations using Gaussian Elimination Method.

$$3x_1 + x_2 + 7x_3 + 2x_4 = 13$$

$$2x_1 - 4x_2 + 14x_3 - x_4 = -10$$

$$5x_1 + 11x_2 - 7x_3 + 8x_4 = 59$$

$$2x_1 + 5x_2 - 4x_3 - 3x_4 = 39$$

$$[A|B] = \left[\begin{array}{cccc|c} 3 & 1 & 7 & 2 & 13 \\ 2 & -4 & 14 & -1 & -10 \\ 5 & 11 & -7 & 8 & 59 \\ 2 & 5 & -4 & -3 & 39 \end{array}\right]$$

$$R_1 \to \frac{1}{3}R_1$$

$$\left[\begin{array}{cccc|c} 1 & 1/3 & 7/3 & 2/3 & 13/3 \\ 2 & -4 & 14 & -1 & -10 \\ 5 & 11 & -7 & 8 & 59 \\ 2 & 5 & -4 & -3 & 39 \end{array}\right]$$

$$R_2 \to R_2 - 2R_1$$

$$R_3 \to R_3 - 5R_1$$

$$R_4 \to R_4 - 2R_1$$

$$\left[\begin{array}{cccc|c} 1 & 1/3 & 7/3 & 2/3 & 13/3 \\ 0 & -14/3 & 28/3 & -7/3 & -56/3 \\ 0 & 28/3 & -56/3 & 14/3 & 112/3 \\ 0 & 13/3 & -26/3 & -13/3 & 91/3 \end{array}\right]$$

$$R_2 \to -\frac{3}{14}R_2$$

$$\left[\begin{array}{cccc|c} 1 & 1/3 & 7/3 & 2/3 & 13/3 \\ 0 & 1 & -2 & 1/2 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -13/2 & 13 \end{array}\right]$$

$$R_3 \Leftrightarrow R_4$$

$$\left[\begin{array}{cccc|c} 1 & 1/3 & 7/3 & 2/3 & 13/3 \\ 0 & 1 & -2 & 1/2 & 4 \\ 0 & 0 & 0 & -13/2 & 13 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$R_3 \to -\frac{2}{13} R_3$$

$$\left[\begin{array}{cccc|c} 1 & 1/3 & 7/3 & 2/3 & 13/3 \\ 0 & 1 & -2 & 1/2 & 4 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

linear system corresponding to the above row echelon matrix

$$x_1 + \tfrac{1}{3}x_2 + \tfrac{7}{3}x_3 + \tfrac{2}{3}x_4 = \tfrac{13}{3}$$

$$x_2 - 2x_3 + \tfrac{1}{2}x_4 = 4$$

$$x_4 = -2$$

$$0 = 0$$

We can ignore the last equation,  since it provides us with no information regarding the values of the variables. The third equation gives $x_4 = -2$. Column 3 of the matrix is a no-pivot column, so $x_3$ is the independent variable. Therefore, we can let $x_3$ take on any value, say $x_3 = c$. Substituting this into the second equation yields

$$x_2 - 2c + \tfrac{1}{2}(-2) = 4 \Rightarrow x_2 = 5 + 2c$$

Then from the first equation, we obtain

$x_1 + \tfrac{1}{3}(5 + 2c) + \tfrac{7}{3}(c) + \tfrac{2}{3}(-2) = \tfrac{13}{3} \Rightarrow x_1 = 4 - 3c$

Therefore, the complete solution set is $\{(4 - 3c, 5 + 2c, c, -2) : c \epsilon \mathbb{R}\}$

Particular solutions can be found by choosing values for $c$. For example, choosing $c = 1$, yields

$x_1 = 1,\ x_2 = 7,\ x_3 = 1$ and $x_4 = -2$.

**Example 7. 6** Solve the following system of linear equations by Gaussian Elimination Method using Scilab.

$$2x + 3z = 1$$

$$y + z = 2$$

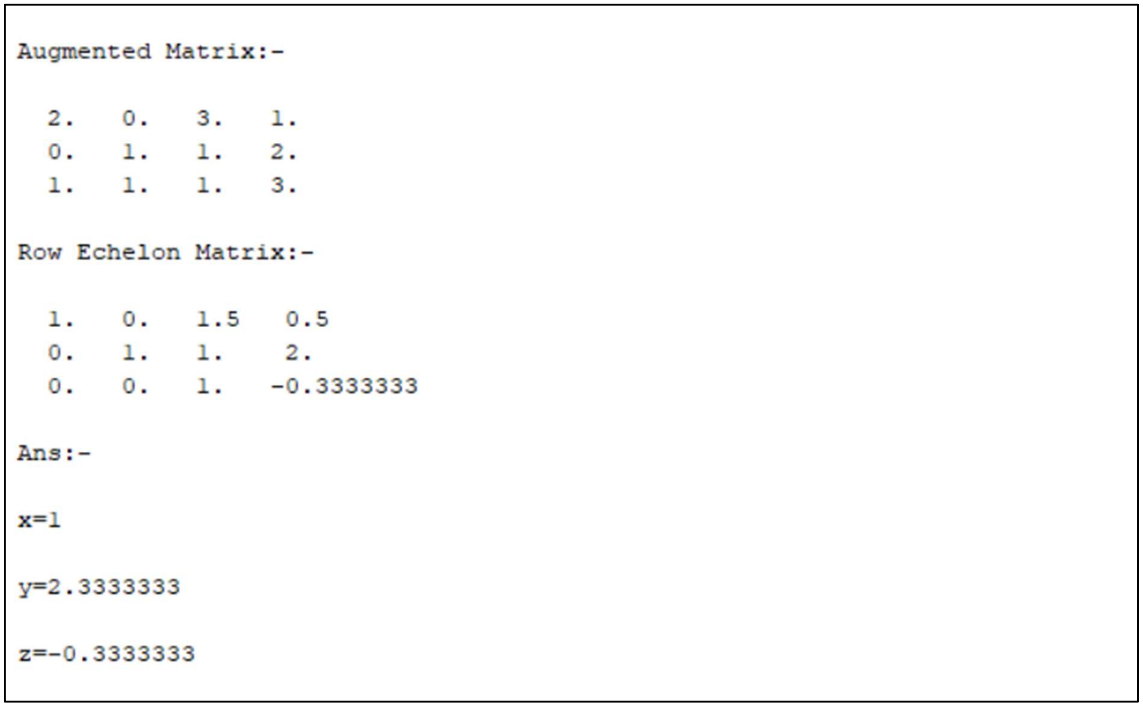
$$x + y + z = 3$$

$$[A|B] = \left[\begin{array}{ccc|c} 2 & 0 & 3 & 1 \\ 0 & 1 & 1 & 2 \\ 1 & 1 & 1 & 3 \end{array}\right]$$

**Scilab code 7.1**

```
//Example 7.6
clc
clear
C=[2 0 3 1;0 1 1 2;1 1 1 3]//C is augmented Matrix
disp("Augmented Matrix:-")
disp(C)
for i=1:3//Elementary Row operation
   for k=(i+1):3
    d=C(i,i)
      t=C(k,i)
      for j=1:4
         C(i,j)=C(i,j)/d
         C(k,j)=C(k,j)-t*C(i,j)
      end
      end


  if (i==3) then


     C(3,4)=C(3,4)/C(3,3)
     C(3,3)=1
     end

end
  disp("Row Echelon Matrix:-")
disp(C)
for i=3:-1:1//back substitution
   x(i,1)=C(i,4)
   for j=(i+1):3
      x(i,1)=x(i,1)-C(i,j)*x(j,1)
end
end
disp('Ans:-')
disp('x='+string(x(1,1))+'')
disp('y='+string(x(2,1))+'')
disp('z='+string(x(3,1))+'')
```

**Console window 7.1**



```
Augmented Matrix:-

  2.   0.   3.   1.
  0.   1.   1.   2.
  1.   1.   1.   3.

Row Echelon Matrix:-

  1.   0.   1.5   0.5
  0.   1.   1.    2.
  0.   0.   1.   -0.3333333

Ans:-

x=1

y=2.3333333

z=-0.3333333
```

## 7. 2 Gauss-Jordan Elimination Method

Let's consider $AX = B$ linear system, the steps for solving $AX = B$ by Gauss-Jordan Elimination Method is given below:

1. Write the augmented matrix $[A|B]$ of the system.

2. Use the elementary row operations to reduce the augmented matrix $[A|B]$ to a matrix $[C|D]$ in reduced row echelon form.

3. Write the linear system corresponding to the reduced row echelon matrix $[C|D]$.

Types of Solution

(a) Inconsistent solution – If any row of the form $[0 \quad 0 \quad \cdots \quad 0|c]$ occur at any stage of the Gaussian -Jordan Elimination method.

(b) Unique solution- when every column of $C$ is pivot column.

(c) Infinitely many solution-when non-pivot column are present in $C$. The variable corresponding to non-pivot column are called independent that can take on any real values, while those corresponding to pivot columns are called dependent variable the values of the dependent variable are then determined in term of independent variable.

**Example 7.7** Solve the following system of linear equations using the Gauss-Jordan Method:

$$x + 2y + 3z = 6$$

$$2x - 3y + 2z = 14$$

$$3x + y - z = -2$$

$$[A|B] = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 2 & -3 & 2 & 14 \\ 3 & 1 & -1 & -2 \end{array}\right]$$

$$R_2 \to R_2 - 2R_1$$

$$R_3 \to R_3 - 3R_1$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & -7 & -4 & 2 \\ 0 & -5 & -10 & -20 \end{array}\right]$$

$$R_2 \Leftrightarrow R_3$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & -5 & -10 & -20 \\ 0 & -7 & -4 & 2 \end{array}\right]$$

$$R_2 \to -\frac{1}{5}R_2$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & 1 & 2 & 4 \\ 0 & -7 & -4 & 2 \end{array}\right]$$

$$R_1 \to R_1 - 2R_2$$

$$R_3 \to R_3 + 7R_2$$

$$\left[\begin{array}{ccc|c} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 4 \\ 0 & 0 & 10 & 30 \end{array}\right]$$

$$R_3 \to \frac{1}{10}R_3$$

$$\left[\begin{array}{ccc|c} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 4 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

$$R_1 \to R_1 + R_3$$

$$R_2 \to R_2 - 2R_3$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

The last matrix is in reduced row echelon form. The linear system corresponding to this final matrix is:

$x = 1,\ y = -2$ and $z = 3.$

**Example 7.8** Solve the following system of linear equations by Gauss-Jordan Elimination Method using Scilab

$$x + y + z = 3$$

$$2x + 3y + 7z = 0$$

$$x + 3y - 2z = 17$$

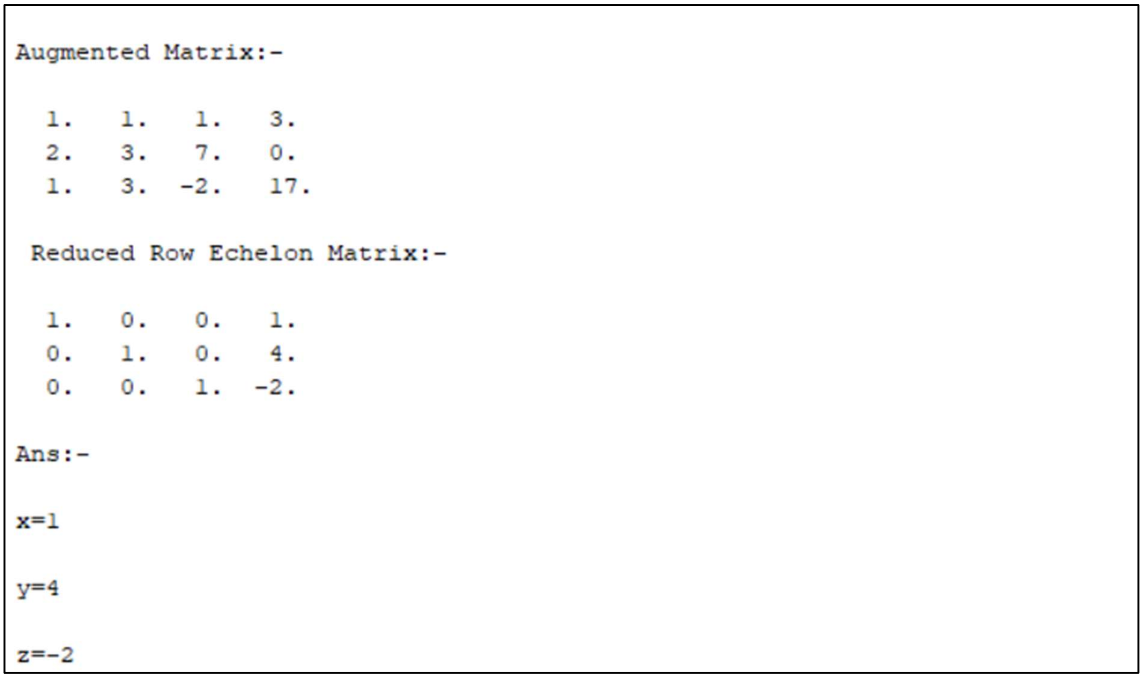
$$[A|B] = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 2 & 3 & 7 & 0 \\ 1 & 3 & -2 & 17 \end{array}\right]$$

**Scilab code 7.2**

```
//Example 7.8
clc
clear
C=[1 1 1 3;2 3 7 0;1 3 -2 17]//C is augmented Matrix
disp("Augmented Matrix:-")
disp(C)
for i=1:3// for row echelon matrix
    for k=(i+1):3
     d=C(i,i)
        t=C(k,i)
        for j=1:4
            C(i,j)=C(i,j)/d
            C(k,j)=C(k,j)-t*C(i,j)
        end
        end
  if (i==3) then
      C(3,4)=C(3,4)/C(3,3)
      C(3,3)=1
      end
 end
for i=3:-1:1// for reduced row echelon matrix
    for k=(i-1):-1:1
```

```
      t=C(k,i)
      for j=1:4
  C(k,j)=C(k,j)-t*C(i,j)
      end
   end
end
disp(" Reduced Row Echelon Matrix:-")
disp(C)
disp('Ans:-')
disp('x='+string(C(1,4))+'')
disp('y='+string(C(2,4))+'')
disp('z='+string(C(3,4))+'')
```

**Console window 7.2**



```
Augmented Matrix:-

   1.   1.   1.   3.
   2.   3.   7.   0.
   1.   3.  -2.   17.

 Reduced Row Echelon Matrix:-

   1.   0.   0.   1.
   0.   1.   0.   4.
   0.   0.   1.  -2.

Ans:-

x=1

y=4

z=-2
```

## 7. 3 Jacobi Method

The Jacobi method is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations ($A$ must be strictly diagonally dominant matrix for system of linear equation $AX = B$).

A square matrix is said to be **diagonally dominant** if, for every row of the matrix, the magnitude of the diagonal entry in a row is larger than or equal to the sum of the magnitudes of all the other (non-diagonal) entries in that row.

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| \leq |a_{ii}|, i = 1,2,3 \dots, n$$

In case of **strictly diagonally dominant** matrix.

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| < |a_{ii}|, i = 1,2,3 \dots, n$$

Let $AX = B$ is square system of $n$ linear equation. $A$ can be decomposed such that

$$A = D - C$$

Where $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}, B = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, C = \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & 0 & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{bmatrix}$$

Now, $AX = B$ can be Re-written as

$(D - C)X = B$

$$DX = B + CX \tag{7.2}$$

Diagonal system of linear equation can be easily solved E.g.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

From above system of linear equation:

$x_1 = \frac{b_1}{a_{11}}, x_2 = \frac{b_2}{a_{22}}, x_3 = \frac{b_3}{a_{33}}$.

The right-hand side vector of (7.2) cannot be a known quantity because it involves the unknown vector $X$. Rather, if we choose (arbitrarily) some specific value for $X$, say $X = X^{(0)}$, then the resulting system $DX = CX^{(0)} + B$ can be readily solved. Let us call the solution of this system as $X^{(1)}$. That is, $DX^{(1)} = CX^{(0)} + B$.Now taking $X = X^{(1)}$ on the right-hand side of (7.2) we can obtain the solution of this system, which we denote as $X^{(2)}$ and repeat this procedure to get a general iterative procedure as

$$DX^{(k+1)} = CX^{(k)} + B, k = 0,1,2,\cdots \qquad (7.3)$$

If $D$ is invertible, then the above iterative procedure can be written as

$$X^{(k+1)} = bX^{(k)} + c, k = 0,1,2,\cdots \qquad (7.4)$$

where $b = D^{-1}C$ and $c = D^{-1}B$.

(7.3) can be Re-written as

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}), i = 1,2,3, ...$$

The iterative procedure (7.4) is called the **Jacobi method**.

The convergence of (7.3) to the solution of $AX = B$, only guaranteed when $A$ is strictly diagonally dominant matrix.

**Example 7.9** Solve the following system of linear equations by the Jacobi method.

$$2x_1 + x_2 = 11$$

$$5x_1 + 7x_2 = 13$$

$A = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}, B = \begin{bmatrix} 11 \\ 13 \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$A$ is strictly diagonally dominant.

If $A$ is decomposed into $D$ and $-C$, $A = D - C$

Then $D = \begin{bmatrix} 2 & 0 \\ 0 & 7 \end{bmatrix}, C = \begin{bmatrix} 0 & -1 \\ -5 & 0 \end{bmatrix}$

$(D - C)X = B$

$DX = B + CX$

$\begin{bmatrix} 2 & 0 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 11 \\ 13 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ -5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$2x_1 = 11 - x_2$

$7x_2 = 13 - 5x_1$

$x_1^{(k+1)} = \frac{11}{2} - \frac{1}{2} x_2^{(k)}$

$x_2^{(k+1)} = \frac{13}{7} - \frac{5}{7} x_1^{(k)}$ $, k = 0,1,2, \ldots$

Choosing initial guess $X^{(0)} = (1,1)$

$X^{(1)}$= (5, 1.1428571)

$X^{(2)}$= (4.9285714, -1.7142857)

$X^{(3)}$= (6.3571429, -1.6632653)

$X^{(4)}$= (6.3316327, -2.68367350)

$X^{(5)}$= (6.8418367, -2.6654519)

$X^{(6)}$= (6.8327259, -3.0298834)

$X^{(7)}$= (7.0149417, -3.0233757)

$X^{(8)}$= (7.0116878, -3.1535298)

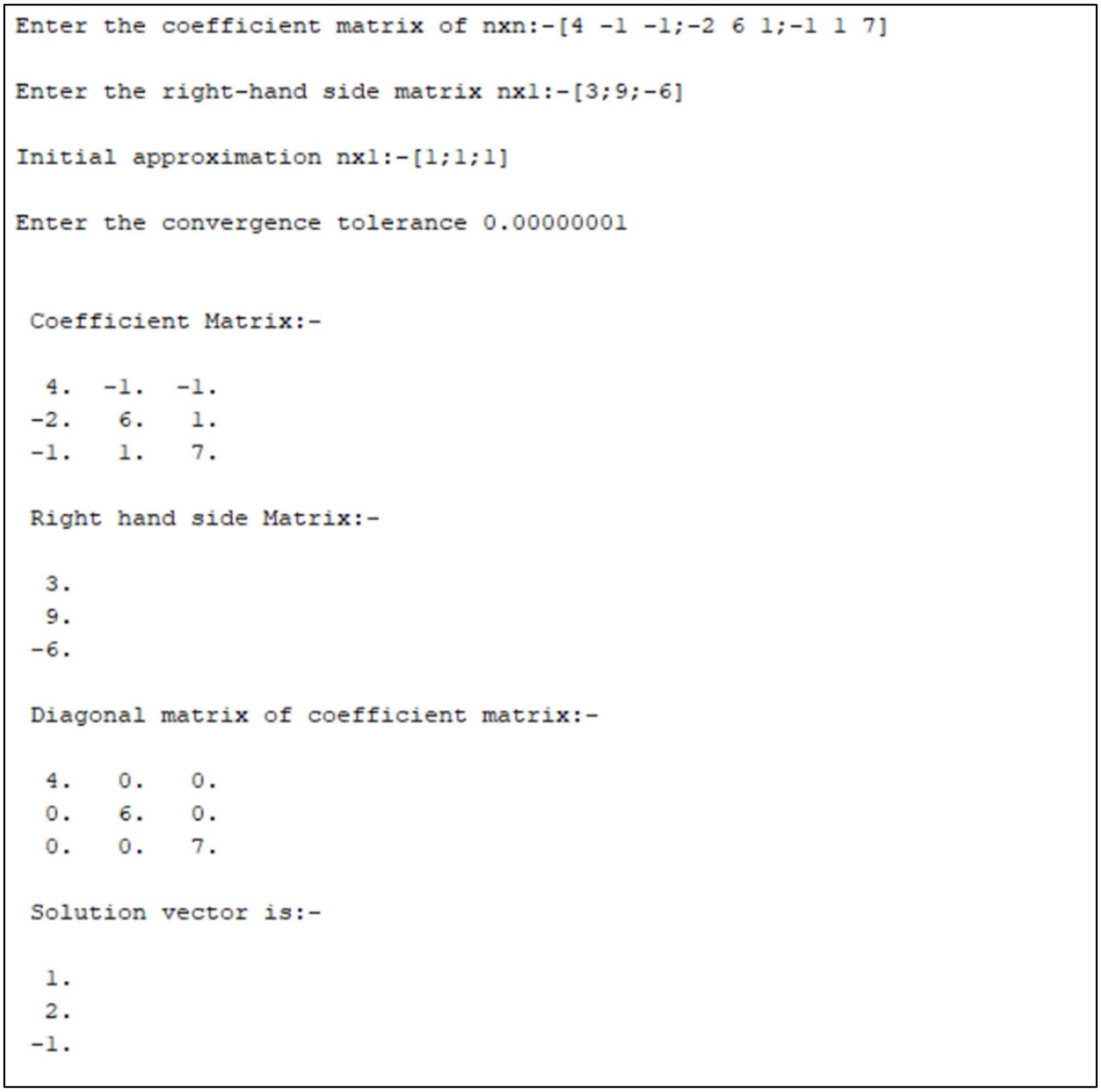
**Solving system of linear equations by Jacobi method using Scilab.**

**Scilab Code 7. 3**

```
//Jacobi Method
clc
clear
A=input(" Enter the coefficient matrix of nxn:-")
B=input(" Enter the right-hand side matrix nx1:-")
X0=input(" Initial approximation nx1:-")
tol=input(" Enter the convergence tolerance ")//Infinity norm of (A*X1-B)
t1=0
t2=0
disp(" Coefficient Matrix:-")
disp(A)
disp(" Right hand side Matrix:-")
disp(B)
```

```
[m, n] = size( A )
  if ( m ~= n )
    disp( " The input matrix is not square")
    t1=1
  end
for i=1:m//To check if the matrix is diagonally dominant
     s=0
     for j=1:m
       s=s+abs(A(i,j))
     end
  if 2*abs(A(i,i))<s then
     disp(" the matrix is not strictly diagonally dominant")
     t2=1
     break
  end
end
if (t1==1|t2==1) then
   disp(" Execute again and enter square and strictly diagonally dominant
coeffiecient matrix ")
   else
for i=1:m
   for j=1:m
     if(i==j) then
        D(i,j)=A(i,j)
     else
        D(i,j)=0
        end
end
end
disp(" Diagonal matrix of coefficient matrix:-")
disp(D)
C=D-A
//Implements the Jacobian algorithm
   Error=1
   while tol<Error
   X1=(inv(D)*C*X0)+(inv(D)*B)//Computes the jacobian updates
   Error=norm((A*X1)-B,'inf')
   X0=X1
end
disp(" Solution vector is:-")
disp(X1)
end
```

**Console window 7. 3**



```
Enter the coefficient matrix of nxn:-[4 -1 -1;-2 6 1;-1 1 7]

Enter the right-hand side matrix nx1:-[3;9;-6]

Initial approximation nx1:-[1;1;1]

Enter the convergence tolerance 0.00000001


 Coefficient Matrix:-

  4.  -1.  -1.
 -2.   6.   1.
 -1.   1.   7.

 Right hand side Matrix:-

  3.
  9.
 -6.

 Diagonal matrix of coefficient matrix:-

  4.   0.   0.
  0.   6.   0.
  0.   0.   7.

 Solution vector is:-

  1.
  2.
 -1.
```

## 7. 4 Gauss-Seidel Method

The Gauss-Seidel Method is an iterative algorithm for determining the solutions of a strictly diagonally dominant or positive definite system of linear equations. ($A$ must be strictly diagonally dominant or positive definite matrix for system of linear equation $AX = B$). Let $AX = B$ is square system of $n$ linear equation. $A$ can be decomposed such that

> A symmetric $n \times n$, real Matrix $A$ is said to be **positive definite** if the scalar $z^T Az$ is strictly positive for every non-zero column vector $z$ of $n$ real numbers. Here $z^T$ denotes the transpose of $z$.

$$A = L + U$$

Where $L$ is lower triangular matrix and $U$ is strictly upper triangular matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}, B = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

$$L = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Now, $AX = B$ can be Re-written as

$(L + U)X = B$

$LX = B - UX$

if we choose (arbitrarily) some specific value for $X$, say $X = X^{(0)}$. Like, we have done in Jacobi method then the resulting system $LX = B - UX$ can be solved using forward substitution for $X^{(1)}$.

$$X^{(k+1)} = L^{-1}(B - UX^{(k)}), k = 0,1,2,\cdots \tag{7.5}$$

The procedure (7.5) is generally continued until the changes made by an iteration are below some tolerance. The iterative procedure $(7.5)$ is called the **Gauss-Seidel Method**.

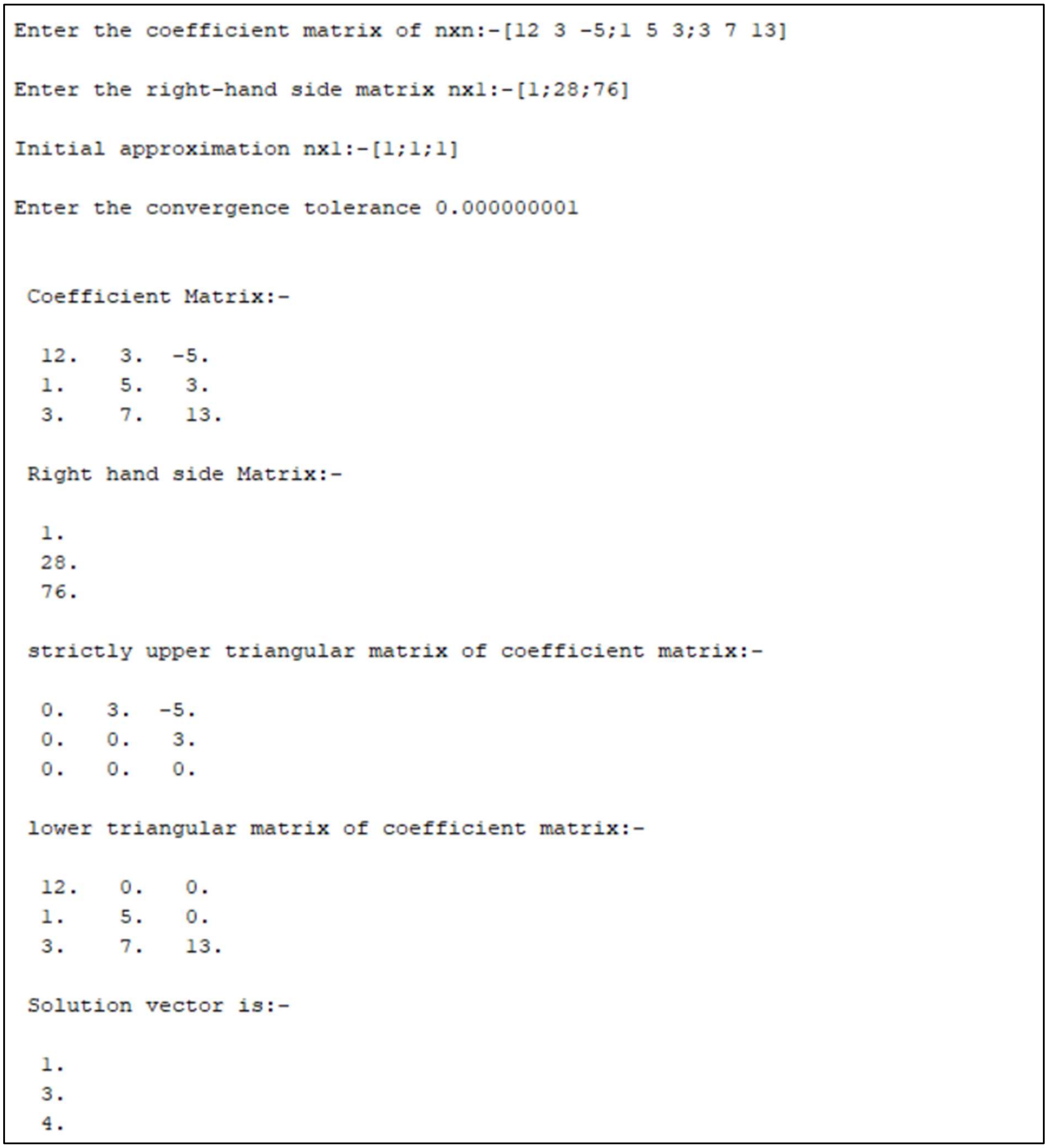
**Solving system of linear equations by Gauss-Seidel Method using Scilab.**

**Scilab code 7.4**

```
//Gauss-Seidel Method
clc
clear
A=input(" Enter the coefficient matrix of nxn:-")
B=input(" Enter the right-hand side matrix nx1:-")
X0=input(" Initial approximation nx1:-")
tol=input(" Enter the convergence tolerance ")//tol>Infinity norm of (A*X1-B)
t1=0
t2=0
disp(" Coefficient Matrix:-")
disp(A)
disp(" Right hand side Matrix:-")
disp(B)
[m, n] = size( A )
```

```
  if ( m ~= n )
    disp( " The input matrix is not square")
    t1=1
  end
for i=1:m//To check if the matrix is diagonally dominant
     s=0
     for j=1:m
        s=s+abs(A(i,j))
     end
  if 2*abs(A(i,i))<s then
     disp(" the matrix is not strictly diagonally dominant")
     t2=1
     break
  end
end
if (t1==1|t2==1) then
   disp(" Execute again and enter square and strictly diagonally dominant
coefficient matrix ")
   else
for i=1:m
   for j=1:m
      if(i<j) then
         U(i,j)=A(i,j)
      else
         U(i,j)=0
         end
end
end
disp(" strictly upper triangular matrix of coefficient matrix:-")
disp(U)
L=A-U
disp(" lower triangular matrix of coefficient matrix:-")
disp(L)
//Implements the Gauss-Seidel  algorithm
   Error=1
   while tol<Error
   X1=inv(L)*(B-(U*X0))//Computes the Gauss-Seidel updates
   Error=norm((A*X1)-B,'inf')
   X0=X1
   end
disp(" Solution vector is:-")
disp(X1)
end
```

## Console window 7.4



```
Enter the coefficient matrix of nxn:-[12 3 -5;1 5 3;3 7 13]

Enter the right-hand side matrix nx1:-[1;28;76]

Initial approximation nx1:-[1;1;1]

Enter the convergence tolerance 0.000000001


 Coefficient Matrix:-

  12.    3.   -5.
  1.     5.    3.
  3.     7.    13.

 Right hand side Matrix:-

  1.
  28.
  76.

 strictly upper triangular matrix of coefficient matrix:-

  0.    3.   -5.
  0.    0.    3.
  0.    0.    0.

 lower triangular matrix of coefficient matrix:-

  12.    0.    0.
  1.     5.    0.
  3.     7.    13.

 Solution vector is:-

  1.
  3.
  4.
```

# CHAPTER **8**

# Finite Difference Method

Chapter-6 was about how to solve ODE numerically. This Chapter deals with solving PDEs numerically. There are many methods for solving PDEs numerically out of which only finite difference method is discussed in this chapter.

**PDE** (Partial Differential Equation) is an equation that contains at least two independent variables, a dependent variable or the unknown function and one or more partial derivative of the unknown function.

In this chapter, a subscript notation is used for PDE such that

$u_x = \frac{\partial u}{\partial x}, u_{xx} = \frac{\partial^2 u}{\partial x^2}, u_{xy} = \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right), u_{yx} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}\right) \dots$

## 8. 1 Classification of PDE

**General terminology of PDE**

The **order** of a PDE is the order of the highest partial derivative which appears in the equation.

E.g. $u_{tt} = c^2 u_{xx}$ is a second-order PDE.

The most general first-order PDE in two independent variables $x$ and $y$ is of the form

$F(x, y, u, u_x, u_y) = 0$

**Linearity of PDE**

A PDE is said to be **linear** if it is of the first degree in the unknown function $u$ and its partial derivatives. Otherwise, it is called **non-linear.**

E.g. $xu_x + yu_y = u$ is linear and $x^2 {u_x}^2 + y^2 {u_y}^2 = u^2$ is non-linear PDE.

The most general first-order linear partial differential equation in two independent variables $x$ and $y$ has the form

$$a(x, y)u_x + b(x, y)u_y + c(x, y)u = d(x, y)$$

Where,$a$, $b$, $c$ and $d$ in general are function of $x$ and $y$.

Similarly, A second-order linear partial differential equation in two independent variables $x$ and $y$ has the form

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$$

Where $A$,$B$,$C$,$D$,$E$,$F$,$G$ are the function of $x$ and $y$.

**Classification of second-order linear PDE**

$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$ is said to be hyperbolic, parabolic or elliptic at point $(x_0, y_0)$ according as the discriminant

$\varDelta(x_0, y_0) \ = B^2(x_0, y_0) \ - 4A(x_0, y_0)\ C(x_0, y_0)$ is positive, zero or negative respectively. If this is true at all point then the equation is said to be hyperbolic, parabolic or elliptic in a domain.

E.g. **1. Tricomi Equation**

$u_{xx} + xu_{yy} = 0$

$A = 1, B = 0, C = x$

$\varDelta > 0$ for $x < 0$ and $\varDelta < 0$ for $x > 0$. So, the Tricomi equation is hyperbolic for $x < 0$ and Elliptic for $x > 0$.

**2. 2D Laplace's Equation**

$u_{xx} + u_{yy} = 0$

$A = 1, B = 0, C = 1$

$\varDelta < 0$ for entire $\mathbb{R}^2$. So, Laplace's Equation is elliptic for entire $\mathbb{R}^2$.

**3. 1D Heat Equation**

$u_t = \alpha^2 u_{xx}$

$A = \alpha^2, B = 0, C = 0$

$\varDelta = 0$ for entire $\mathbb{R}^2$. So, 1D Heat Equation is parabolic for entire $\mathbb{R}^2$.

**4. 1D Wave Equation**

$u_{tt} = c^2 u_{xx}$

$A = c^2, B = 0, C = -1$

$\varDelta > 0$ for entire $\mathbb{R}^2$. So, 1D Wave Equation is hyperbolic for entire $\mathbb{R}^2$.

**Homogeneous and non-homogeneous linear PDE**

A linear PDE is said to be homogeneous if each of its term contains either $u$ or one of its partial derivatives. Otherwise, it is called non-homogeneous.

E.g. $u_{tt} = c^2(u_{xx} + u_{yy})$ is homogeneous and $u_{xx} + u_{yy} = f(x,y)$ is non-homogeneous.

A function of independent variables of PDE that satisfy PDE in entire space of independent variables is called **Solution of PDE.**

## 8. 2 Finite Difference Fundamentals

The finite difference method (FDM) works by replacing the region over which the independent variables in the PDE are defined by a finite grid (produced by discretization of independent variables region) of points at which the dependent variable is approximated. The partial derivatives in the PDE at each grid point are approximated from neighboring values by using Taylor's theorem.

**Taylor's Theorem**

Let's $u(x)$have $n$ continuous derivative's over the interval $(a, b)$. Then for

$a < x_0$ , $x_0 + h < b$,

$$u(x_0 + h) = u(x_0) + hu_x(x_0) + \frac{h^2 u_{xx}(x_0)}{2!} + \cdots \frac{h^{n-1}u_{n-1}(x_0)}{(n-1)!} + o(h^n) \tag{8.1}$$

Where,

$$u_x = \frac{du}{dx}, u_{xx} = \frac{d^2u}{dx^2}, \dots, u_{n-1} = \frac{d^{n-1}u}{dx^{n-1}}$$

The usual interpretation of Taylor's theorem says that if we know the value of $u$ and the values of its derivatives at point $x_0$ then we can write down the equation (8.1) for its value at the (nearby) point $x_0 + h$. This expression contains an unknown quantity(error) which is written in as $o(h^n)$ and pronounced 'order $h$ to the $n$'. If we discard the term $o(h^n)$ in (8.1) ($i.e.$ truncate the right-hand side of (8.1)) we get an approximation to $u(x_0 + h)$. The error in this approximation is $o(h^n)$.

Let $f: [a, b] \to \mathbb{R}, f', f'', \dots, f^{(n-1)}$ be continuous on $[a, b]$ and suppose $f^n$ exist on $(a, b)$. Then there exist $c \in (a, b)$ such that

$$f(b) = f(a) + f'(a)(b-a) + \cdots + \frac{f^{(n-1)}(a)(b-a)^{n-1}}{(n-1)!} + \frac{\boldsymbol{f^n(c)(b-a)^n}}{\boldsymbol{n}!}$$

In the (8.1) both $x_0$ and $x_0 + h$ are grid points for FDM. This allows us to rearrange equation (8.1) to get so-called Finite Difference (FD) approximations to derivatives which have $o(h^n)$ errors.

Let's see simple finite difference approximation to derivative, Truncating (8.1) after the first derivative

$$u(x_0 + h) = u(x_0) + hu_x(x_0) + o(h^2)$$

$$u_x(x_0) = \frac{u(x_0+h) - u(x_0)}{h} + \frac{o(h^2)}{h}$$

$$= \frac{u(x_0+h) - u(x_0)}{h} + o(h)$$

Neglecting the error term $o(h)$.

$$u_x(x_0) \approx \frac{u(x_0+h) - u(x_0)}{h} \tag{8.2}$$

(8.2) is called a first order finite difference approximation to $u_x(x_0)$ since the approximation error $= o(h)$which depends on the first power of $h$. This approximation is called a forward finite difference approximation since we start at $x_0$ and step forwards to the point $x_0 + h$. $h$ is called the step size ($h > 0$).

The following simple example shows that our forward difference approximation (8.2) works and has the stated order of accuracy. We choose a simple function for $u$. Let $u(x) = x^2$. We will find the first order forward FD approximation to $u_x(3)$ using step $size\ h = 0.1$

Using (8.2)

$$u_x(3) \approx \frac{(3+0.1)^2 - (3)^2}{0.1} = 6.1$$

The exact answer from basic Calculus is clearly $u_x(3) = 6$ so the error in the approximation is 6.1 – $6 = 0.1$. Repeating the problem with $h = 0.05$ ($i.e.$ half the step size) gives,

$$u_x(3) \approx \frac{(3+0.05)^2 - (3)^2}{0.05} = 6.05$$

The error is 6.05 – $6 = 0.05$. The approximation formula (8.2) is first order so the errors should be proportional to $h$ which is seen to be the case: halving the step size results in a halving of the error.

**Constructing a Finite Difference Toolkit**

We now construct common FD approximations to common partial derivatives. For simplicity, we suppose that $u$ is a function of only two variables, $x$ and $t$. We will approximate the partial derivatives of $u$ with respect to $x$. As $t$ is held constant $u$ is effectively a function of the single variable $x$ so we can use Taylor's formula (8.1) where the ordinary derivative terms are now partial derivatives and the arguments are $(x, t)$ instead of $x$. Finally, we will replace the step size $h$ by $\Delta x$ so that (8.1) becomes,

$$u(x_0 + \Delta x, t) = u(x_0, t) + \Delta x u_x(x_0, t) + \cdots \frac{\Delta x^{n-1} u_{n-1}(x_0,t)}{(n-1)!} \quad + o(\Delta x^n) \qquad (8.3)$$

Truncating (8.3) to $o(\Delta x^2)$ gives,

$$u(x_0 + \Delta x, t) = u(x_0, t) + \Delta x u_x(x_0, t) + o(\Delta x^2) \qquad (8.4)$$

Now we derive some FD approximations to partial derivatives. Rearranging (8.4) gives,

$$\begin{aligned} u_x(x_0, t) &= \frac{u(x_0 + \Delta x, t) - u(x_0, t)}{\Delta x} + \frac{o(\Delta x^2)}{\Delta x} \\ &= \frac{u(x_0 + \Delta x, t) - u(x_0, t)}{\Delta x} + o(\Delta x) \qquad (8.5) \end{aligned}$$

Equation (8.5) holds at any point $(x_0, t)$. In numerical schemes for solving PDEs, we are restricted to a grid of discrete $x$ values, $x_1, x_2, \ldots, x_n$ and discrete $t$ levels $t_1, t_2, \ldots$ We will assume a constant grid spacing, $\Delta x$ in $x$ so that $x_{j+1} = x_j + \Delta x$ Evaluating Equation (8.5) for a point,$(x_j, t_k)$ on the grid gives,

$$u_x(x_j, t_k) = \frac{u(x_{j+1}, t_k) - u(x_j, t_k)}{\Delta x} + o(\Delta x) \qquad (8.6)$$

We will use the common subscript/superscript notation,

$$u_j^k = u(x_j, t_k)$$

So that dropping $o(\Delta x)$ from (8.6) becomes

$$u_x(x_j, t_k) = \frac{u_{j+1}^k - u_j^k}{\Delta x} \tag{8.7}$$

(8.7) is the first order forward difference approximation to $u_x(x_j, t_k)$ that we derive previously approximation (8.2). We now derive another FD approximation to $u_x(x_j, t_k)$. Replacing $\Delta x$ by $-\Delta x$ in (8.4) gives,

$$u(x_0 - \Delta x, t) = u(x_0, t) - \Delta x u_x(x_0, t) + o(\Delta x^2) \tag{8.8}$$

Evaluating (8.8) at $(x_j, t_k)$ and rearranging as previously gives,

$$u_x(x_j, t_k) = \frac{u_j^k - u_{j-1}^k}{\Delta x} \tag{8.9}$$

(8.9) is the first order backward difference approximation to $u_x(x_j, t_k)$.

Our first two FD approximations are first order in $x$ but we can increase the order (and so make the approximation more accurate) by taking more terms in the Taylor series as follows. Truncating (8.3) to $o(\Delta x^3)$,then replacing $\Delta x$ by $-\Delta x$ and subtracting this new expression from (8.3) and evaluating at $(x_j, t_k)$. gives, after some algebra,

$$u_x(x_j, t_k) = \frac{u_{j+1}^k - u_{j-1}^k}{2\Delta x} \tag{8.10}$$

(8.10) is called the second order central difference FD approximation to $u_x(x_j, t_k)$.

We could construct even higher order FD approximations to $u_x$ by taking even more terms in the Taylor series but we will stop at second order approximations to first order derivatives.

Many PDEs of interest contain second order (and higher) partial derivatives so we need to derive approximations to them. We will restrict our attention to second order unmixed partial derivatives $i.e.\ u_{xx}$.

Truncating (8.3) to $o(\Delta x^4)$ gives,

$$u(x_0 + \Delta x, t) = u(x_0, t) + \Delta x u_x(x_0, t) + \frac{\Delta x^2 u_{xx}(x_0, t)}{2!} + \frac{\Delta x^3 u_{xxx}(x_0, t)}{3!} + o(\Delta x^4) \tag{8.11}$$

Replacing $\Delta x$ by $-\Delta x$ in (8.11)

$$u(x_0 - \Delta x, t) = u(x_0, t) - \Delta x u_x(x_0, t) + \frac{\Delta x^2 u_{xx}(x_0, t)}{2!} - \frac{\Delta x^3 u_{xxx}(x_0, t)}{3!} + o(\Delta x^4) \tag{8.12}$$

Adding (8.11) and (8.12) gives

$$u(x_0 + \Delta x, t) + u(x_0 - \Delta x, t) = 2u(x_0, t) + \Delta x^2 u_{xx}(x_0, t) + o(\Delta x^4) \tag{8.13}$$

Evaluating (8.13) at $(x_j, t_k)$ and using our discrete notation gives,

$$u_{j+1}^k + u_{j-1}^k = 2u_j^k + \Delta x^2 u_{xx}(x_j, t_k) + o(\Delta x^4) \tag{8.14}$$

Rearranging (8.14) and dropping the $o(\Delta x^4)$ error term gives,

$$u_{xx}(x_j, t_k) = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2} \tag{8.15}$$

(8.15) is the second order symmetric difference FD approximation to $u_{xx}(x_j, t_k)$. These results are put into Table 8.1 to form a FD approximation toolkit. FD approximations to partial derivatives with respect to $t$ are derived in a similar manner and are included in Table 8.1.

| partial derivative | finite difference approximation | type | order |
|---|---|---|---|
| $\frac{\partial u}{\partial x} = u_x(x_j, t_k)$ | $\frac{u_{j+1}^k - u_j^k}{\Delta x}$ | forward | first in $x$ |
| $\frac{\partial u}{\partial x} = u_x(x_j, t_k)$ | $\frac{u_j^k - u_{j-1}^k}{\Delta x}$ | backward | first in $x$ |
| $\frac{\partial u}{\partial x} = u_x(x_j, t_k)$ | $\frac{u_{j+1}^k - u_{j-1}^k}{2\Delta x}$ | central | second in $x$ |
| $\frac{\partial^2 u}{\partial x^2} = u_{xx}(x_j, t_k)$ | $\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2}$ | symmetric | second in $x$ |
| $\frac{\partial u}{\partial t} = u_t(x_j, t_k)$ | $\frac{u_j^{k+1} - u_j^k}{\Delta t}$ | forward | first in $t$ |
| $\frac{\partial u}{\partial t} = u_t(x_j, t_k)$ | $\frac{u_j^k - u_j^{k-1}}{\Delta t}$ | backward | first in $t$ |
| $\frac{\partial u}{\partial t} = u_t(x_j, t_k)$ | $\frac{u_j^{k+1} - u_j^{k-1}}{2\Delta t}$ | central | second in $t$ |
| $\frac{\partial^2 u}{\partial t^2} = u_{tt}(x_j, t_k)$ | $\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\Delta t^2}$ | symmetric | Second in $t$ |

**Table 8.1**

## 8. 3 2D Laplace's Equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad (8.16)$$

Let's take computational domain rectangular. The computational domain is discretized using constant grid $\Delta x$ and $\Delta y$ in $x$ and $y$ direction such that $x_1, x_2, \ldots, x_M$ and $y_1, y_2, \ldots, y_N$. $u(x_i, y_j)$ is denoted by $u_{i,j}$.

The boundary condition is given for (8.16). So, $(M-2) \times (N-2)$ grid points are unknown out of $M \times N$ grid points.

Constructing a Finite difference scheme for (8.16) using Table 8.1

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0 \qquad (8.17)$$

Taking $\Delta x = \Delta y$ (8.17) becomes

$$u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{4} \tag{8.18}$$

(8.18) is our FDS for Laplace's equation.

(8.18) can be solved using Jacobi iteration by taking some initial guess for interior grid points such that

$$u_{i,j}^{m+1} = \frac{u_{i-1,j}^{m} + u_{i+1,j}^{m} + u_{i,j-1}^{m} + u_{i,j+1}^{m}}{4} \tag{8.19}$$

Superscript $m$ is iteration index, iterates (8.19) till $|u^{m+1} - u^{m}|_\infty < tol.$

$tol$ is pre-defined tolerance (allowed error).

$|A|_\infty$ is infinity norm of matrix $A$. It is equal to maximum of absolute sum of row. $i.e|A|_\infty = max \sum_{j=1}^{n}|a_{ij}|$ for all $i$.

Let's understand (8.19) for a simple example.

The rectangular computational domain is given below

| | | | | |
|---|---|---|---|---|
| 8.9 | 8.9 | 8.9 | 8.9 | 8.9 |
| 8.4 | $u_{2,3}$ | $u_{3,3}$ | $u_{4,3}$ | 9.2 |
| 7.2 | $u_{2,2}$ | $u_{3,2}$ | $u_{4,2}$ | 9.4 |
| 6.1 | 6.8 | 7.7 | 8.7 | 9.8 |

Setting all interior grid points to zero as the initial guess

$m = 0$

$$u_{2,2}^{1} = \frac{u_{1,2}^{0} + u_{3,2}^{0} + u_{2,1}^{0} + u_{2,3}^{0}}{4}$$

$$u_{2,2}^{1} = \frac{7.2 + 0 + 6.8 + 0}{4} = 3.5$$

$$u_{3,2}^{1} = \frac{u_{2,2}^{0} + u_{4,2}^{0} + u_{3,1}^{0} + u_{3,3}^{0}}{4}$$

$$u_{3,2}^{1} = \frac{0 + 0 + 7.7 + 0}{4} = 1.925$$

$$u_{4,2}^{1} = \frac{u_{3,2}^{0} + u_{5,2}^{0} + u_{4,1}^{0} + u_{4,3}^{0}}{4}$$

$$u_{4,2}^{1} = \frac{0 + 9.4 + 8.7 + 0}{4} = 4.525$$

$$u_{2,3}^{1}=\frac{u_{1,3}^{0}+u_{3,3}^{0}+u_{2,2}^{0}+u_{2,4}^{0}}{4}$$

$$u_{2,3}^{1}=\frac{8.4+0+0+8.9}{4}=4.325$$

$$u_{3,3}^{1}=\frac{u_{2,3}^{0}+u_{4,3}^{0}+u_{3,2}^{0}+u_{3,4}^{0}}{4}$$

$$u_{3,3}^{1}=\frac{0+0+0+8.9}{4}=2.225$$

$$u_{4,3}^{1}=\frac{u_{3,3}^{0}+u_{5,3}^{0}+u_{4,2}^{0}+u_{4,4}^{0}}{4}$$

$$u_{4,3}^{1}=\frac{0+9.2+0+8.9}{4}=4.525$$

Do the same for $m=1,2,3,\ldots M$ until $|u^{M}-u^{M-1}|_{\infty}<tol$

The solution is $u^{M}$.

**Example 8. 1** Find the potential in square plate $0\leq x\leq 2,\ 0\leq y\leq 2$ using Scilab if the upper side is kept at potential $1000\,sin\left(\frac{\pi}{2}x\right)$ and the other sides are grounded.

No charge is lying on the square plate so,

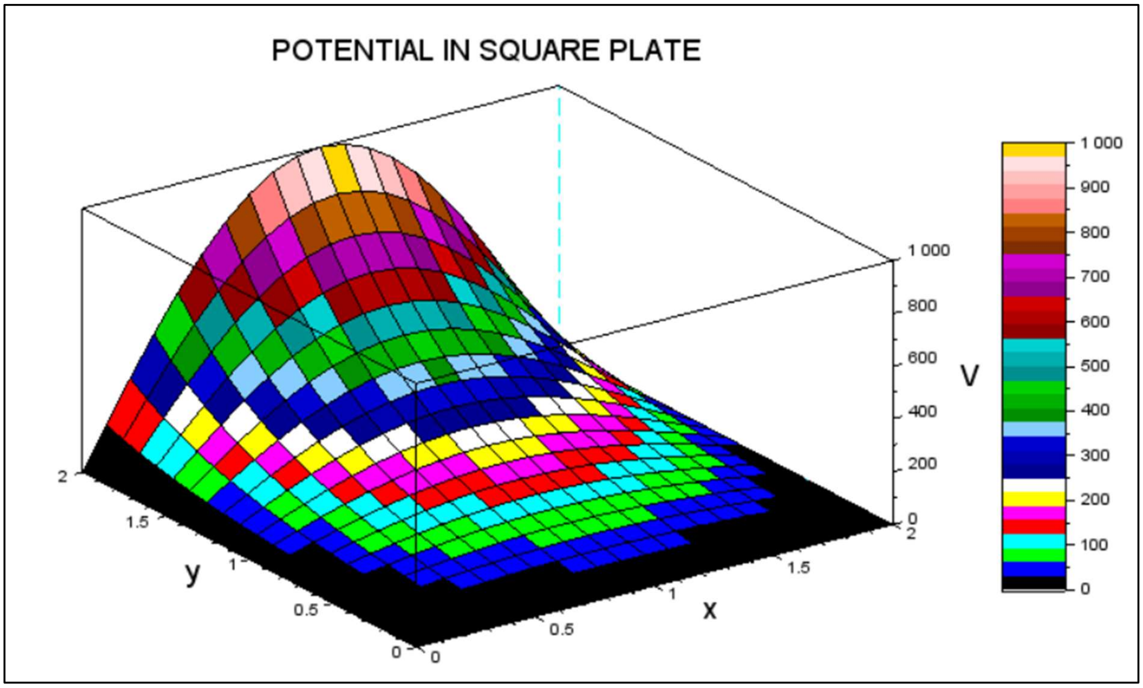
$\frac{\partial^2 V}{\partial x^2}+\frac{\partial^2 V}{\partial y^2}=0, V$ is potential

**Scilab code 8.1**

```
//Laplace's equation
clc
clear
clf
dx=0.1//step size in x direction
dy=0.1//step size in y direction
nx=21
ny=21
x=[0:0.1:2]
y=[0:0.1:2]
tol=0.00000000000001
//matrix u store grid points values such that at u(i,j)=u(xi,yj)
//applying boundary condition and taking intial guess zero for interior grid
points
for i=1:nx
```

```
   for j=1:ny
      if j==ny then
         u(i,j)=1000*sin(%pi*0.5*(i-1)*dx)

        else
           u(i,j)=0

      end
end
end
err=1//Initialization of while loop
uk=u
while err>tol//jacobi iteration
   for i=2:nx-1//both for do the jacobian updates to entire grid points except for boundary
      for j=2:ny-1
         uk(i,j)=(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1))*0.25

      end
   end
   err=norm( uk - u, 'inf' )//infinity norm of |uk-u|
   u=uk

end
//plotting potential of square plate
surf(x,y,uk','color_flag',0)
colorbar(0,1000)
xlabel('x','Fontsize',4)
ylabel('y','Fontsize',4)
zlabel('V','Fontsize',4)
title('POTENTIAL IN SQUARE PLATE','Fontsize',4)
```

**Graphic window 8.1**



## 8. 4 1D Heat Equation

Consider the following initial-boundary value problem for the heat equation

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2}, 0 < x < L \qquad (8.20)$$

BC: $u(0,t) \;=\; 0 \; u(L,t) \;=\; 0$

IC: $u(x,0) \;=\; f(x)$

Discretizing space and time such that $(x_1, x_2, \dots, x_N), (t_1, t_2, \dots, t_K)$ using constant grid $\Delta x$ and $\Delta t$.

Using Table 8.1

$$u_t(x_n, t_k) = \frac{u_n^{k+1} - u_n^k}{\Delta t} \qquad (8.21)$$

$$u_{xx}(x_n, t_k) = \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} \qquad (8.22)$$

Substituting (8.21) and (8.22) in (8.20)

$$\frac{u_n^{k+1} - u_n^k}{\Delta t} = \alpha^2 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2}$$

$$u_n^{k+1} = u_n^k + \frac{\alpha^2 \Delta t}{\Delta x^2}(u_{n+1}^k - 2u_n^k + u_{n-1}^k) \tag{8.23}$$

(8.23) is our Finite difference scheme for the 1D heat equation.

**Stability of finite difference scheme for the heat equation**

A finite difference scheme is stable if the errors made at one time step of the calculation do not cause the errors to be magnified as the computations are continued. A neutrally stable scheme is one in which errors remain constant as the computations are carried forward. If the errors decay and eventually damp out, the numerical scheme is said to be stable. If, on the contrary, the errors grow with time the numerical scheme is said to be unstable.

Let $u_n^k = \phi_k e^{in\Delta x\theta}$ then

Substituting $\phi_k e^{in\Delta x\theta}$ into (8.23)

$$\phi_{k+1} e^{in\Delta x\theta} = \phi_k e^{in\Delta x\theta} + \frac{\alpha^2 \Delta t}{\Delta x^2}\left(e^{i\Delta x\theta} - 2 + e^{-i\Delta x\theta}\right)\phi_k e^{in\Delta x\theta}$$

$$\phi_k e^{in\Delta x\theta} + \frac{\alpha^2 \Delta t}{\Delta x^2}(2\cos(\Delta x\theta) - 2)\phi_k e^{in\Delta x\theta}$$

$$\phi_{k+1} = \phi_k - \frac{\alpha^2 \Delta t}{\Delta x^2} 4\sin^2(\Delta x\theta/2)\phi_k$$

$$\phi_{k+1} = (1 - \frac{\alpha^2 \Delta t}{\Delta x^2} 4\sin^2(\Delta x\theta/2))\phi_k$$

Now for stability, we require that $|\phi_{k+1}| \le |\phi_k|$ so that

$$|1 - \frac{\alpha^2 \Delta t}{\Delta x^2} 4\sin^2(\Delta x\theta/2)\,| \le 1$$

$$-2 \le -\frac{\alpha^2 \Delta t}{\Delta x^2} 4\sin^2(\Delta x\theta/2) \le 0$$

The right inequality is satisfied automatically, while the left inequality can be re-written in the form:

$$\frac{\alpha^2 \Delta t}{\Delta x^2} 4\sin^2(\Delta x\theta/2) \le 2$$

Since $\sin^2(\Delta x\theta/2) \le 1$this condition is satisfied for all $\theta$ provided

$$\Delta t \le \frac{\Delta x^2}{2\alpha^2}$$

**Example 8.2** Consider a laterally insulated metal bar of length 1 and such that $\alpha^2 = 1$ in the heat equation. Suppose that the ends of the bar are kept at temperature 0°C and initially the temperature along the bar is given by $sin(\pi x)$ find the temperature in $0 \le t \le 0.2$ using Scilab.

Using $u_n^{k+1} = u_n^k + \frac{\alpha^2 \Delta t}{\Delta x^2}(u_{n+1}^k - 2u_n^k + u_{n-1}^k)$

$\alpha^2 = 1$ taking $\Delta t = 0.004$ and $\Delta x = 0.1$

$\frac{2\alpha^2 \Delta t}{\Delta x^2} = 0.8 \le 1$ so (8.23) is stable

Using initial and boundary condition

$u(x_1, t_k) = \ u(x_{11}, t_k) = 0$ for all time level

$u(x_n, t_1) = sin(\pi(n-1)\Delta x)$ along the entire rod.

$$u_n^{k+1} = u_n^k + 0.4(u_{n+1}^k - 2u_n^k + u_{n-1}^k)$$

$k = 1$

$$u_2^2 = u_2^1 + 0.4(u_3^1 - 2u_2^1 + u_1^1)$$

$$u_3^2 = u_3^1 + 0.4(u_4^1 - 2u_3^1 + u_2^1)$$

$$\vdots$$

$$u_{10}^2 = u_{10}^1 + 0.4(u_{11}^1 - 2u_{10}^1 + u_9^1)$$

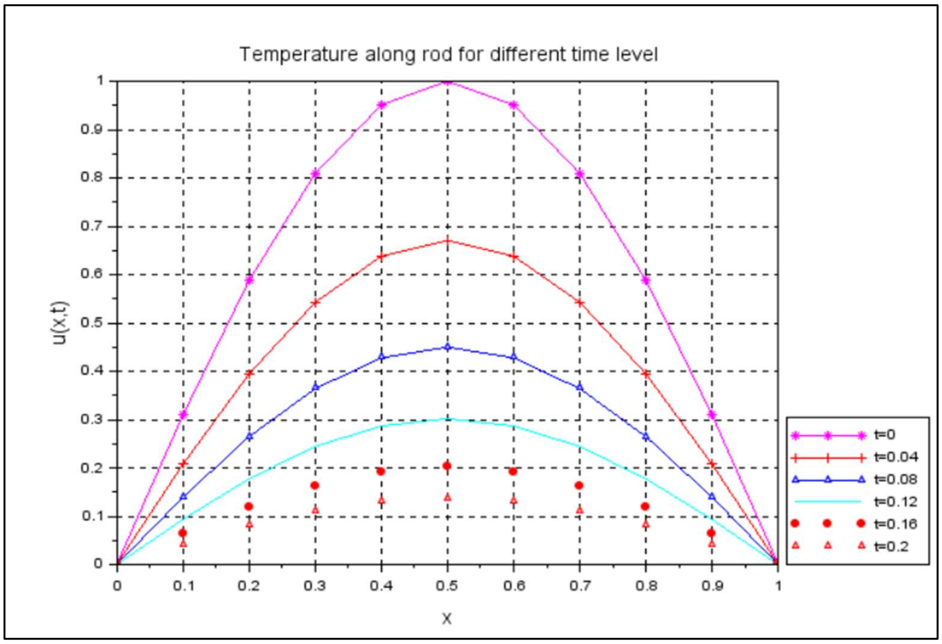
Similarly, for $k = 2,3, \dots ,50$

**Scilab code 8.2**

```
//solving 1D heat equation
clc
clear
clf
x=[0:0.1:1]'
//u(n,k)
//u(space,time)
dx=0.1//step size in x
dt=0.004//step size in t
nx=11//number of space grid poits
nt=51//number of time grid points
```

```
//using initial and boundary condition
u(1,:)=0
u(11,:)=0
u(:,1)=sin(%pi*(0:10)*dx)
//using FDS
for n=2:10
   u(n,2)=u(n,1)+0.4*(u(n+1,1)-2*u(n,1)+u(n-1,1))
end
for k=2:50
   for n=2:10
      u(n,k+1)=u(n,k)+0.4*(u(n+1,k)-2*u(n,k)+u(n-1,k))
end
end
plot(x,u(:,1),'m-*-')
plot(x,u(:,11),'r-+-')
plot(x,u(:,21),'b-^-')
plot(x,u(:,31),'c-')
plot(x,u(:,41),'r.')
plot(x,u(:,51),'r^')
xgrid(4)
xlabel('x','Fontsize',3)
ylabel('u(x,t)','Fontsize',3)
title('Temperature along rod for different time level','Fontsize',3)
legend('t=0 ','t=0.04 ','t=0.08 ','t=0.12 ','t=0.16 ','t=0.2 ',-4)
```

**Graphic window 8.2**



## 8. 5 1D Wave Equation

Consider the following initial boundary value problem for the Wave Equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, 0 < x < L \tag{8.24}$$

$$\text{BC: } u(0,t) \ = \ 0 \ u(L,t) \ = \ 0$$

$$\text{IC: } u(x,0) \ = \ f(x)$$

$$\frac{\partial u}{\partial t}(x,0) = g(x) \tag{8.25}$$

Discretizing space and time such that $(x_1, x_2, \dots, x_N), (t_1, t_2, \dots, t_K)$ using constant grid $\Delta x$ and $\Delta t$.

$$u_{tt}(x_n, t_k) = \frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} \tag{8.26}$$

$$u_{xx}(x_n, t_k) = \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} \tag{8.27}$$

Substituting (8.25) and (8.26) in (8.24)

$$\frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} = c^2 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2}$$

$$u_n^{k+1} = 2u_n^k - u_n^{k-1} + (c\Delta t/\Delta x)^2 (u_{n+1}^k - 2u_n^k + u_{n-1}^k)$$

$$u_n^{k+1} = r^2 u_{n+1}^k + 2(1 - r^2) u_n^k + r^2 u_{n-1}^k - u_n^{k-1} \tag{8.28}$$

(8.28) is FDS for 1D Wave equation where $r \ = \ (c\Delta t/\Delta x)$ is known as the Courant Number. We observe that the Discrete Equation (8.28) involves three distinct levels in which known data is transferred from steps $k - 1$ and $k$ to step $k \ + \ 1$.

(8.28) is stable only when $r \leq 1$. Detail analysis of the stability of (8.28) is not discussed here.

**Initial Conditions - Starting the Solution**

The 3-level scheme poses some challenges when imposing the initial conditions. If we imagine a row of false mesh points at time $t = -\Delta t = t_0$, then the initial velocity condition (8.25) can be approximated using central differences as:

$$\frac{u_n^2 - u_n^0}{2\Delta t} = g(x_n)$$

$$u_n^0 = u_n^2 - 2\Delta t g(x_n) \tag{8.29}$$

(8.28) when $k = 1$

$$u_n^2 = r^2 u_{n+1}^1 + 2(1 - r^2)u_n^1 + r^2 u_{n-1}^1 - u_n^0 \tag{8.30}$$

Substituting (8.29) in (8.30) for 2nd $(t_2)$ time level

$$u_n^2 = \frac{1}{2}(r^2 u_{n+1}^1 + 2(1 - r^2)u_n^1 + r^2 u_{n-1}^1) + \Delta t g(x_n) \tag{8.31}$$

BC, IC, (8.28), and (8.31) are sufficient for knowing $u$ at all-time levels.

**Example 8. 3** Solve 1D wave equation such that $c = 1$ . Initial and boundary conditions are given below:

$$\text{BC: } u(0,t) \ = \ 0 \ u(L,t) \ = \ 0$$

$$\text{IC: } u(x,0) \ = \ sin(3\pi x/10)$$

$$\frac{\partial u}{\partial t}(x,0) = 0$$

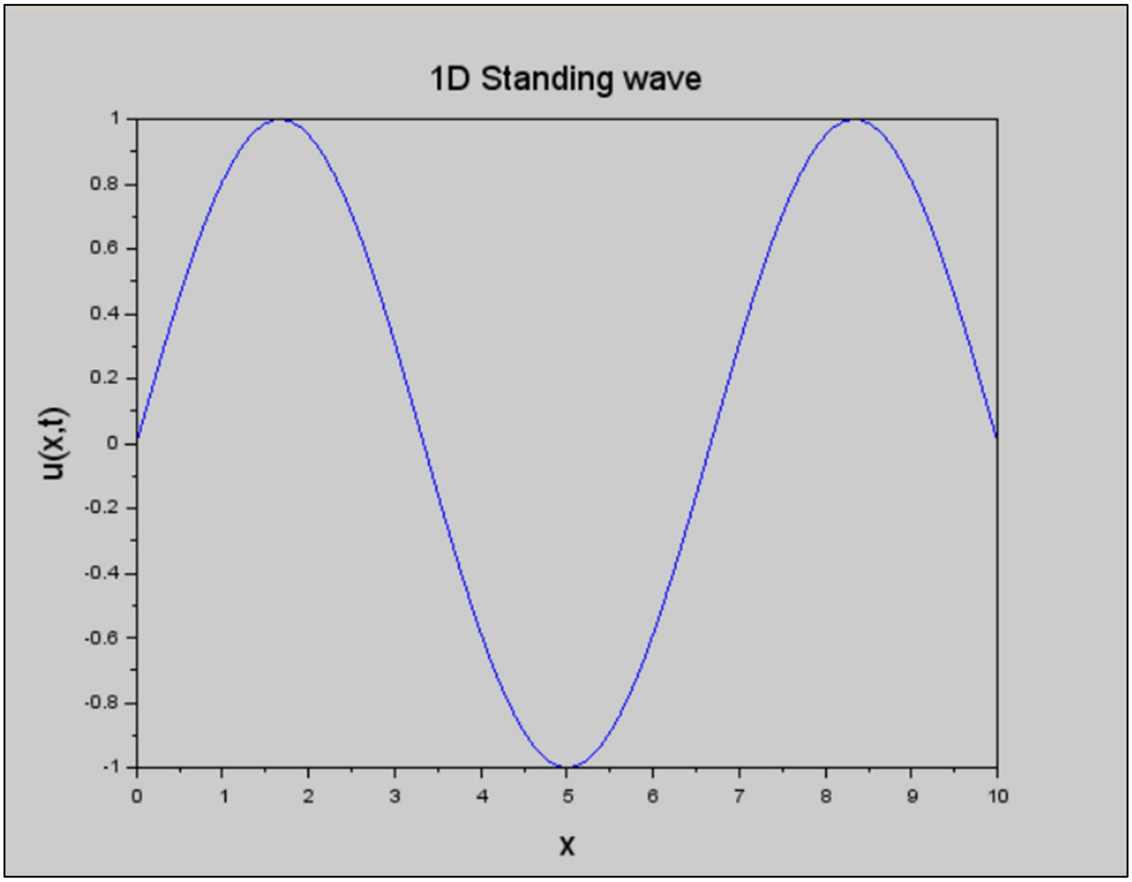
Consider $\Delta t = 0.05$ ,$\Delta x = 0.1$. So that $r = 0.5 \leq 1$ for the stability of (8.28).

**Scilab code 8. 3**

```
//1D Wave Equation
clc
clear
//u(space,time)
//u(n,k)
//u(101,1201)
x(:,1)=[0:0.1:10]
u=zeros(101,1201)
dx=0.1//step size in x
dt=0.05//step size in t
c=1
r=0.5//Courant Number
function z=f(x)
  z=sin(3*%pi*x/10)
endfunction
//using initial and boundary condition
u(1,:)=0
```

```
u(101,:)=0
u(:,1)=f((0:100)*dx)
for n=2:100
   u(n,2)=0.5*(r^2*u(n+1,1)+2*(1-r^2)*u(n,1)+r^2*u(n-1,1))
end
for k=2:1200
   for n=2:100
      u(n,k+1)=r^2*u(n+1,k)+2*(1-r^2)*u(n,k)+r^2*u(n-1,k)-u(n,k-1)
      end
end
realtimeinit(0.05)
figure(1)
realtime(0)
plot(x,u(:,1))
xlabel("x","Fontsize",4)
ylabel("u(x,t)","Fontsize",4)
title('1D Standing wave',"Fontsize",4)
h_compound = gce()
h_axes = gca()
h_axes.data_bounds = [0,-1;10,1]
for i=1:1200
   realtime(i)
 drawlater()
   h_compound.children.data = [x,u(:,i+1)]
   drawnow()
end
```

## Graphic window 8. 3



# About the Author

## Rohan Verma

**Academic Qualifications:**

- **M.Sc. Physics**, Indian Institute of Technology (IIT) Kanpur, India
- **B.Sc. Physics**, University of Delhi, India

**Connect with Me:**

- LinkedIn: rohan-verma-0534a71ab
- Email: rohanv.i341@gmail.com